Design Exercise: Distributing 1080p Video in Large Scale Sunrise Systems LED Signs

System architecture and design for upper level video flow control to enable rapid manufacture of high resolution (1080p) LED video signs from existing Sunrise modules.

Contents

1. Problem Statement

In Summer of 2015, Sunrise Systems, Inc. funded GreenArrays to design a simple, practical video distribution system that would enable Sunrise to build very large LED signs, running full 1080p video from an HDMI source, from it existing LED sign modules. Each module had an array of LED pixels with pitches typically 7.62mm; each module was controlled by an ARM processor receiving commands on an RS485 compatible serial interface running 4.5 Megabit asynchronous communications. A full sign might have on the order of 2,000 modules.

Sunrise has kindly given us permission to discuss the resulting design in this App Note. As it turned out, Sunrise concluded that its customers for such signs did not actually require the versatility of displaying real time 1080P video from an arbitrary source, so the design was never produced.

Referenced documents:

http://www.analog.com/media/en/technical-documentation/data-sheets/ADV7800.pdf
https://ez.analog.com/servlet/JiveServlet/download/1548-7-7431/ADV7800%20Manuals.zip
http://www.analog.com/en/search.html?q=ADV7610BBCZ-P-RL
HDMI Specification 1.3a for High Definition Multimedia Interface
CEA-861-D for EDID 1.3 structure specification and CEA Extension Version 3-

This section outlines our thought process in understanding and addressing this problem.

1.1 Top Level Constraints

The sign will be built to a specified display resolution corresponding with a standard video format.

- The target is 1080p; no larger formats are contemplated.
- A given system will be set up for a single video format and all input will be in that format. 1080p is the benchmark.
- No video format conversion is contemplated.
- The display may be built to a smaller size than full 1080p by using a smaller array of modules. When this is done the system must be configurable to "window" a prescribed rectangle (of the same size as the physical display in pixels) from the incoming video image onto the display. So a nonzero X,Y origin must be supported as must a rectangle size X,Y less than or equal to the full size of the incoming video frame.
- Pixel data are in 24-bit RGB format. Sunrise believes that 21 bits may be enough and has stated that 18 bits is not acceptable.

The sign will be built of the following components:

- An incoming video chip. The Analog Devices ADV7800 is postulated as the candidate.
- An array of Sunrise Systems LED sign modules. The communication protocol used to deliver video to these modules may be changed to facilitate a clean system design but no other fundamental changes are to be made in the modules unless they produce a significant architectural advantage.
- A video signal processing and routing system will connect the incoming video chip with the array of modules. This study is primarily concerned with the design and implementation of this subsystem.
- We will probably need to provide a processor to manage the sign as a whole, programming the incoming video chip, supporting test modes, and reporting status to an upstream system.

1.2 Timing

The project had an aggressive schedule for design and prototyping.

1.3 Existing Modules

This section summarizes the specification of the existing functional modules.

1.3.1 Incoming Video Chip

An example of a suitable chip is the Analog Devices ADV7800 which converts standard video formats on standard media to a (24-bit) RGB parallel interface when suitably programmed.

Product page: http://www.analog.com/en/products/audio-video/video-decoders/adv7800.html
Data sheet: http://www.analog.com/media/en/technical-documentation/data-sheets/ADV7800.pdf
HW/SW Design Manuals: https://ez.analog.com/servlet/JiveServlet/download/1548-7-7431/ADV7800%20Manuals.zip

VDD is 1.8V for digital supply, nominally 3.3V for analog and apparently for digital I/O, 2.5V for DDR SDRAM. Further investigation needed, level shifting apparently necessary to drive 1.8v cleanly

Another possibility, costing about 1/3 as much, is the AD 7610. Data may be retrieved from this URL:

http://www.analog.com/en/search.html?q=ADV7610BBCZ-P-RL

Programming is done over one or two I^2C ports. It does not seem to support video format conversion nor windowing.

Output pixel rate will be somewhere between 124.416 MHz (8.04 ns), the *average* rate of 1080p at 60 Hz, and 150 MHz (6.66 ns), the maximum rate specified by the chip. The pixel clock per VESA 12p is 148.5 MHz (6.7 ns); that video format allows for 1125 line times per frame and 2200 pixel times per line, hence the difference between average rate and the sustained rate during each horizontal line. Our calculations assume the chip's max rate of 150 MHz for TPIX and FPIX as being prudent.

Output digital signal drive needs further investigation and is apparently programmable. The normal N drive appears to be on the order of 3.2 mA while the P drive seems to be only 400 µA, which might be appropriate for I^2C which has external pull-up resistors but not for the parallel pixel data out which certainly should not be single ended.

1.3.2 Sunrise LED Modules

As of 14 Sep 2015, the modules Sunrise contemplates using are in several pixel formats and pitches to achieve desired sign dimensions and pixel resolution. No module would be wider than 40 pixels (this upper limit is architecturally important.) LED control is mulltiplexed 2, 4 or 10 ways, but that is not relevant outside the module:

Pixels **PX,PY**	Pitch	mm	Inches	Modules @ 1080p **MX, MY**
32x32	3.75mm	120^2	4.7^2	60x34 (2040)
24x24	6.35mm	152.4^2	6^2	80X45 (3600)
30x16/24/32	7.62mm	228.6 in x	9 x various	64x68/45/34
40x16/24/32	7.62mm	304.8 in x	12 x various	48x68/45/34
32x24	12.7mm	406.4x304.8	16x12	60x34 (1632)

We are assuming that the maximum practical sign array is 60x34 (2,040) modules. The X dimension (60) is architecturally important.

Each module has an ARM processor with three *unidirectional* RS422 interfaces presently doing async at 4.25 or 0.875 Mbit/s but these will be re-purposed. We may change the ARM code (written in Forth) to facilitate protocol changes that would make this system more tractable, and if necessary will arrange for bidirectional RS422. Multiple modules are presently handled by daisy chaining the high speed interface (data, one in & one out) and discretely wiring each module's lower speed control channel. We have schematics and source code listings for existing modules.

1.3.2.1 Serial Communications

The ARM processor has three active UARTs. All external connections are made on 8-pin connectors J6 and J7. These connections are RS422/486 driven by AM26LS31 and received by AM26LS32A, with V_{DD} set at 5V. All modules' ARM chips use a 12.000 MHz crystal controlling internal clock of six times that frequency (to 72.000 MHz) for computer and peripheral clocks. The UARTs according to the code are set by divisors against 1/16 this frequency, or 4.500 MHz. Baud rates described in the code are NOT running at the frequencies stated but rather at the values shown below. All UARTS are set with Line Control Register (LCR) of x03 giving 8 data bits, 1SB, no parity. We do not know yet how tolerant this UART is of inaccurate baud rates.

UART1 (control) receives signal from J7.7/8 and the modules can be configured to repeat this signal out J6.7/8 without any processing, and with or without buffering. This input is stiffened by 1k/1k/1k voltage dividers between V_{DD} and ground, unlike the other inputs. This UART is normally configured in the code for 875 kbps, however that gives a divisor of 5.125 of which 5 is used so the actual frequency appears to be 900.0 kbps.

UART2 can receive signal from RX-L (J7.1/2) and can transmit on TX-R (J6.1/2). UART3 can receive from RX-R (J6.3/6) and can transmit on TX-L (J7.3/6). These inputs have 100Ω parallel termination. The code is configured for a baud rate of 4.375 Mbps, however that gives a divisor of 1.125 so the actual frequency appears to be 4.500 Mbps.

Therefore it appears we have the resources necessary to establish bidirectional communication with the module if the module's code is altered for this application and if we wish to do so.

1.4 Acknowledgments

This work, as noted earlier, was funded by Sunrise Systems. In addition, Sunrise provided LED modules along with their source code and documentation, and fabricated the prototype PCBs.

We licensed SwiftX™ from FORTH, Inc. to compile and flash changes for the Sunrise modules' code.

System design was done primarily by Greg Bailey and Mark Smeder. F18 programming was done by Greg Bailey and Charley Shattuck. Hardware design was done by Mark Smeder, Greg Bailey and Warren Sarkison, with Glenn Sanderson's assistance in solving some signal quality problems on the VCTL prototype. ARM programming was done by Dean Sanderson. FPGA design was done by James Bowman. Prototype assembly was done by Mark Smeder and Greg Bailey. PCB rework was done by Greg Bailey, Mark Smeder and Matt Williams of Sunrise. Greg Bailey did the study and analyses of the ADV7611 chip and of the JTAG programming of FPGAs. Documentation was produced by Greg Bailey. At various points in this project, all of the above team members contributed to each other's activities, and in addition Eric Harrington and Steve Budak of Sunrise contributed insights and answered questions about the problem statement, parts selection and manufacturing issues. Finally, Chuck Moore contributed ideas during the initial stages and served helped us understand wierd things as they came up.

Existing GreenArrays software components, created by our entire team, were used in this project, as was the GA144 polyFORTH® system whose implementation was done by GreenArrays and whose architecture was kindly contributed by FORTH, Inc. This is the first project in which the work done by Stefan Mauerhofer on excitation of high frequency crystals was depended upon, both for the asynchronous communications channels and for the Ethernet interface.

1.5 Video Distribution

The central problem is to capture and distribute the relevant pixel data to each of the display modules with minimal complexity, wiring problems, and wasted energy. (Some energy saving measures may not be desirable. For example, not propagating data thru a daisy chain any farther than it is needed would tend to add a power consumption signal component looking like a 60 Hz saw-tooth.)

Sunrise, and we, are thinking in terms of attaching a GA144 to each module within the cable harness. This would allow each of the GA144s to buffer and move one module's worth of pixel data into the ARM without stress in storage (40*32 or 1280 pixels max, 3840 bytes; This could readily be stored in 60 nodes at one byte per word of local RAM, or 30 nodes at 2 bytes per word) or time (16 ms frame time to move 3840 bytes, on the order of 1.920 Mbit/s).

Given this, everything then depends on getting each module's data into its dedicated GA144 once per frame, using a small fraction of a frame time so that we *do* have most of a frame time to talk with the ARM. To accomplish that we must split the incoming video into multiple streams, each at a significantly longer pixel period than TPIX.

1.5.1 General Strategies

We considered running all pixels down the left side of the display, buffering each module row at one place on the left side and then propagating the data across the row much as we'd discussed for direct LED drive. The vertical bus would be running 24 bits wide at TPIX, presumably daisy chained between devices, but the transmission across each module row could be done on the order of TPIX*MY. This is still 226 ns per pixel (4.41 Mpix/s or 105.9 Mbit/s) and that transmission rate would require a parallel interface of some width going across each module row. Ignoring the logical and electrical problems that come with this high speed parallel transmission, this would also require buffering one module row (32 pixel rows) of data at each of 34 places on the left side. 32 pixel rows is 32x1920 or 61440 pixels, 184320 bytes of buffered data in something (FPGA) out to the left of each module row. That's perhaps 34 expensive FPGAs, one high speed bus down the height of the sign and 34 high speed buses across the module rows. While obvious this strategy has some serious defects and we have concluded it is not a good way to go.

A less obvious strategy is to select, in each video scan line, the pixels destined for each column of modules and run the data down those columns basically in parallel. Unlike the left side, there is no requirement for large buffers but only the PX pixels in a FIFO for each column. The data rate is better than any horizontal strategy because there are fewer data to be sent down a column of modules than across a row of them; each pixel going down a column needs to be disposed of in time on the order of TPIX*MX which is larger than TPIX*MY. MX of 48 gives worst case 1080p timing:

Pixels **PX,PY**	**MX**	Pixel time TPIX*MX ns	Pixel Freq Mpix/s	Bit freq Mbit/s	Bit time ns
32x32	60	399.60	2.5	60.1	16.65
40x32	48	319.68	3.13	75.08	13.32

The pixel clock per VESA 12p is 148.5 MHz (6.7 ns) however that video format allows for 1125 line times and 2200 pixel times per line. The extra vertical lines do not help but the horizontal overhead does; 148.5 1920 2200 */ is 129.6 MHz (7.7 ns), and this is the average rate we have to be able to move down the columns. TPIX and FPIX assume the higher rate of 150 MHz for prudence; to see best case numbers using our preferred strategy, multiply by 771/666 for time and 666/771 for frequency.

We have determined that these rates aren't feasible with 2-wire synchronous protocol. We will have to use a parallel daisy chain, probably of three data bits and a half-frequency clock (triggering on both edges.)

However it is implemented, though, there are two significant compelling advantages to feeding columns rather than rows; first being that there is no need for large buffers, , and second that the data rate will always be lower for normal display aspect ratios whose height is less than width in pixels. Therefore we have concluded that feeding columns of modules is the best way to go.

1.5.2 Methods of Distributing to Columns

Thus far we have thought of two different ways to distribute the pixel stream to the module columns. Each has its own strengths and weaknesses; we will continue investigating both of them until all of the tradeoffs are better understood. Certainly, at least one of them should be able to do the job, if not both.

1.5.2.1 The GA144 Method

The distribution is handled as an additional function of the GA144s in the top several module rows. This assumes a parallel, daisy chained bus running across each such row (12 bits plus clock). Two rows minimum are required to carry 24-bit pixels. However because we cannot keep up with TPIX this will require not just one pair of rows but probably three or four pairs to demultiplex the pixels into a slower stream for a total of MMY modules at the top of each column. Cooperation will be needed between these MMY involved modules in order to drive their data down the column (and up for data destined to the rows above each of them).

Feasibility of this method depends on several things including coordination of the demultiplexing, ability to sample reliably during the short pixel hold time, signal qualilty and skew control over as long a distance as 15.7 inches, and the coordination of traffic up and down the module column. It may turn out necessary to have some separate logic such as an FPGA on the left end to handle the clock stretching and the timing for demultiplexing; however if the GA144 cannot grab a pixel value from the bus driven directly by the incoming video chip some method of latching the 24 data lines three or four ways might be necessary.

Windowing in both X and Y will need to be implemented inside the GA144s, meaning that the topmost pair of rows and the leftmost column will have different code, no surprise because the rows will need different code anyway to control the up/down movement of column data.

Additionally this method gives serious failure modes; failure of a single module in the top MMY rows will affect the rest of the display to its right, and even a transient failure could easily cause gross loss of synchronization. Measures will certainly be needed to make this method robust. At this time we don't think this method is feasible with the G144A12.

1.5.2.2 The FPGA Method

An alternative way to achieve the same thing involves no change in the protocol for driving the pixel data down the columns. It differs only in using a single FPGA rather than several rows of GA144s.

The FPGA would have 24-bit data and clock connected to the incoming video chip, nearby on the same PCB so that signal quality and skew are not issues. Internally, the FPGA would window the pixel stream with counters. Other counters within each scan line would drop pixels into 60 short (40x24, 960-bit) FIFOs, each of which feeds one of 60 drivers for 2-wire synchronous pixel data going down a column. The synchronous bit clock would be derived from the pixel clock so that there is no problem with overrunning these FIFOs.

Architecturally if we wound up needing more than 2-wire down the column we might not have enough GPIOs on the FPGA.

The most obvious electrical problem is adequate drive for the 60 2-wire output signals. Another unique problem for this method is that because all the distribution logic is at one point, the 2-wire signals would have to be carried a fair distance to reach the appropriate modules at the tops of the columns. If the sign control was physically placed at top center of the display, then in worst case we would have a 2-wire sync line running half the width of the sign, or 12 to 31 feet, apparently. At minimum this may require line drivers and good transmission lines for the signals.

In terms of failure modes the FPGA becomes a single point of failure for the whole sign, just like the incoming video chip and the sign control computer. Because these *are* single points of failure the assembly carrying them should most likely be available onsite as a spare, along with other major modules such as power supplies and a few display modules.

1.5.3 Moving Data Down a Column

We've designed and implemented on the bench a method of moving data down a column using a GA144's parallel ports to daisy chain. The parallel data word should work at 3 data bits and might be feasible at 2 data bits; this will be investigated closely because each additional data bit added to reduce data rate increases cost per module, cost of wire, and the burden on the FPGA for pins and signal drive.

1.5.3.1 Inter-GA144 Transmission Protocol

The input to each GA144 consists of n data bits and a clock. The clock is half frequency, meaning that we trigger on each of its edges. The setup time for data relative to clock edge should be positive but we believe may be zero; our method of processing the clock signal has a built-in 7.2 ns delay which gives us a "free" setup time on the data.

Within the GA144, pixel data for its module are captured based upon counting clocks. The start of a scan line and the start of a frame will be recognized by a comparatively long time-interval between clock edges, with the exact criteria to be determined. In the event that a module loses bit/word synchronization during a frame, it will ideally be able to re-synch at the start of the next frame. The capture of data and its processing within the chip will have no effect at all on the transit of data through that chip and out to the next step in the daisy chain.

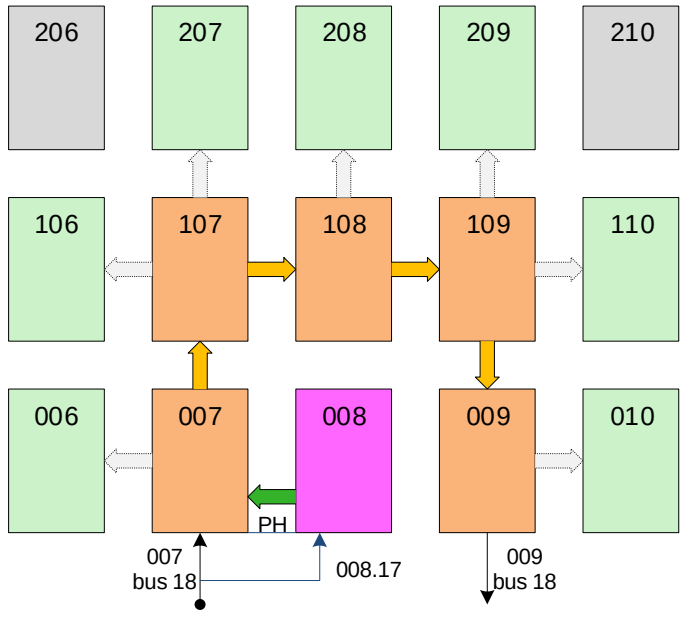
The output of each GA144 consists of n data bits and a clock, all driven at the same time through the parallel output port. Thus in addition to "repeating" the signals at each chip, they are automatically de-skewed.

If it is necessary to generate each of these signals as differential pairs, the GA144 can drive both wires directly from its parallel port. This will increase the delay through the chip but should not impact its bandwidth. However, some sort of differential receiver will need to be built for each line, for good common mode rejection in an environment of high power usage and thus noise on power and, probably, ground.

We have not yet measured our ability to directly receive AC coupled clock and data with reliability; this may become a question for cost control in the case of differential transmission lines. More on this below.

1.5.3.2 Pipeline Implementation within GA144

The data path for parallel reception, transmission and capture is as follows:



Parallel Data Path

Node 008 waits for a clock edge by reading **up** . Writes **io** to change state of phantom wake-up for 007, and sets to wait on other clock phase.

Node 007 reads **up+down** to capture data (and clock) lines on phantom wake-up. Writes **up+down** to send same to 107. Writes **io** to wait on other phantom wake-up phase.

Nodes 107, 108, 109 and 009 are simple wire nodes, moving data from input port to output port(s) at a typical cost of 4.85 ns. Processing may be done along the way to restore inverted pins for differential drive.

Any of the light green nodes may receive a "tap" of incoming data for internal processing. Each is ***absolutely*** required to be punctual, i.e. to ***always*** be reading by the time the next datum is available.

Basic bench code for implementing this path, with no taps active and no differential restoration, is as follows:

```
set which test to compile here
```

```
1032 list
mark 1 compile,
,
code 1036 1038 thru,
,
```

```
1034 list
- load descriptors,
stim 10705 +node 10705 /ram io /b io /a,
..3E 3B 2E 2B 3E 3B 2E 2B 2A 2B,
....10 /stack 0 /p,
10008 +node 10008 /ram 1A /io,
....16 A97 16 A97 16 A97 16 A97 16 A97,
....10 /stack up /b io /a 0 /p,
10007 +node 10007 /ram 0 /io,
....0 800 0 800 0 800 0 800 0 800 10 /stack,
....'-d-u 105 /a io /b 0 /p,
10107 +node 10107 /ram,
....down /a left /b 0 /p,
10108 +node 10107 /ram,
....left /a right /b 0 /p,
10109 +node 10107 /ram,
....right /a down /b 0 /p,
10009 +node 10107 /ram,
....down /a data /b 0 /p,
```

```
1036 list
- signal sim reclaim 10705 node 0 org,
stim 00 begin ! ..,
..01 . .. . .. . .. . .. . .. . .. . ..,
..07 . .. . .. . .. . .. . .. . .. end,
0E reclaim exit
```

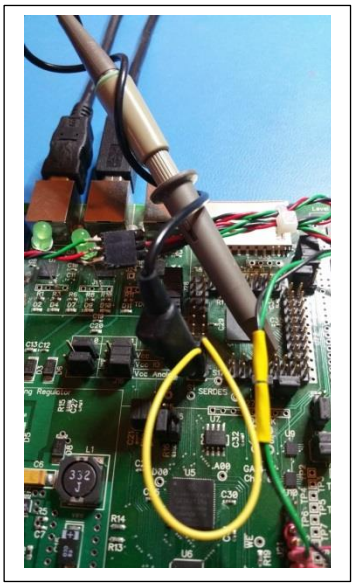
```
node 008 triggers on both clock edges.,
..on each edge sends same signal on phantom,
..wakeups to 007/009 and inverts wake dir.,
..nominal cycle 13ns latency 7.2ns,
,
node 007 reads triggered input from up/down,,
..writes value to up/down to pass it to 107,
..which should always be ready, and inverts,
..wake dir. nominal cyc 16.75ns lat 5-10ns.,
,
nodes 107-108-109-009 wire the data stream,
..from 007 to the node 9 parallel port with,
..taps done by punctual 2-port writes.,
..nominal cyc 13.0ns lat 5-10 ns.
```

```
1038 list
- incoming reclaim 10008 node 0 org,
prx 00 begin begin,
..@b drop ! unext end,
,
02 reclaim 10007 node 0 org parallel in
set 00 -1 1 dup push dup,
....02 push dup push dup push dup push dup,
....04 push dup push dup push push,
run begin 06 @ ! !b unext run ;,
,
08 reclaim 10107 node 0 org path to out,
set 00 -1 1 dup push dup,
....02 push dup push dup push dup push dup,
....04 push dup push dup push push,
..begin begin 06 @ !b unext unext,
07 reclaim exit
```

The 2-port read by node 007 is safe because 107 will never write to it. The 2-port write is safe because the parallel port is configured for input; the instruction will write to the data register but that register does nothing. The code for node 705 is part of the test described in the next section.

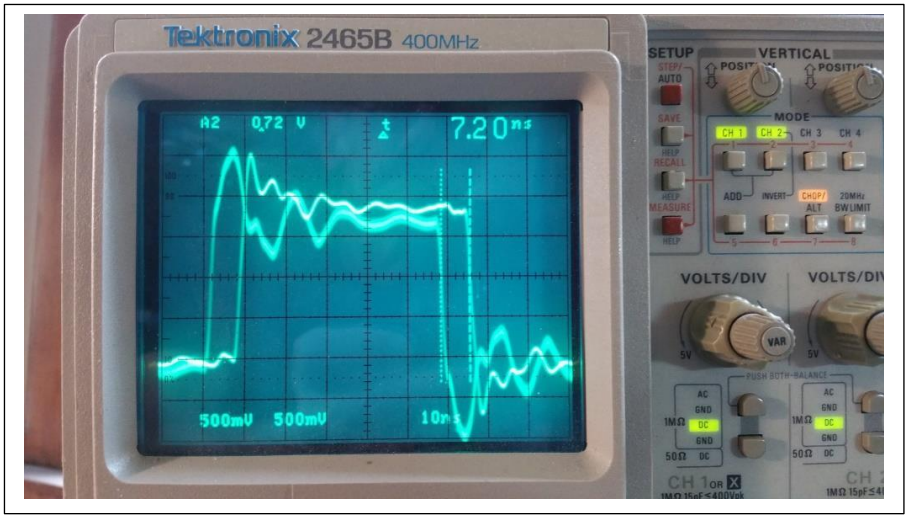
Timings in nodes 008 and 007 have an additional time on the order of 10 ns in one cycle of many, when the **unext** is exhausted. This has not been measured precisely but should not give a problem at current data rates. Precise measurements will be made after we learn whether any additional processing is required within the critical path.

1.5.3.3 Pipeline Testing



Node 705 of the target chip was programmed to produce a relatively low frequency (10 Mwords/sec) stimulus consisting of two data bits and a clock, using the code shown above, and jumpers used to carry its signals to nodes 008 and 007 as shown to the left.

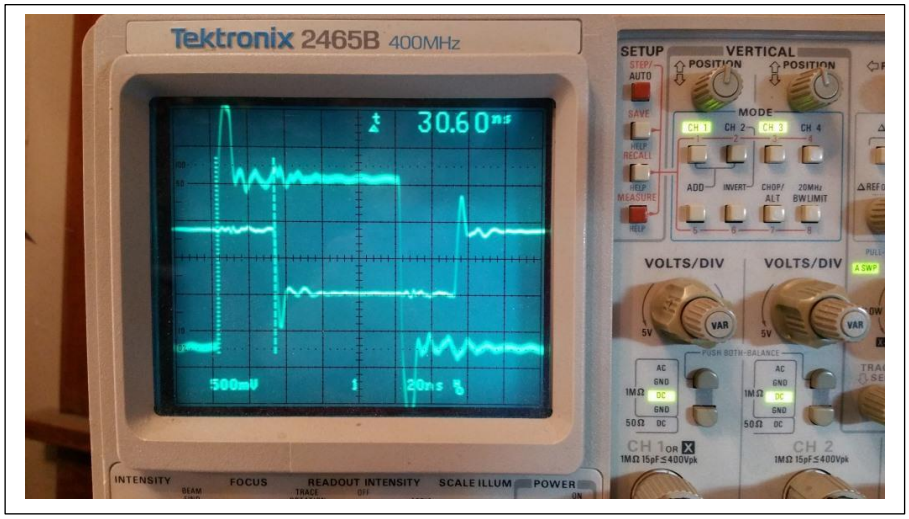
In the trace to the right, we temporarily had node 008 drive one of its output pins to show the delay between incoming clock and setting of **io** to be 7.20 ns. That is also the delay to the trigger for reading and latching node 007's parallel input.



The trace to the right shows the delay between incoming clock signal at pin 008.17 and one of the outgoing data pins driven by node 009.

Subtracting 7.20 ns for the stimulus from node 008 to 007, we have a transit time of (23.40/5) or 4.68 ns per F18, which is the best case behavior of punctual wire nodes.

The system can carry a higher data rate than this delay time, of course.

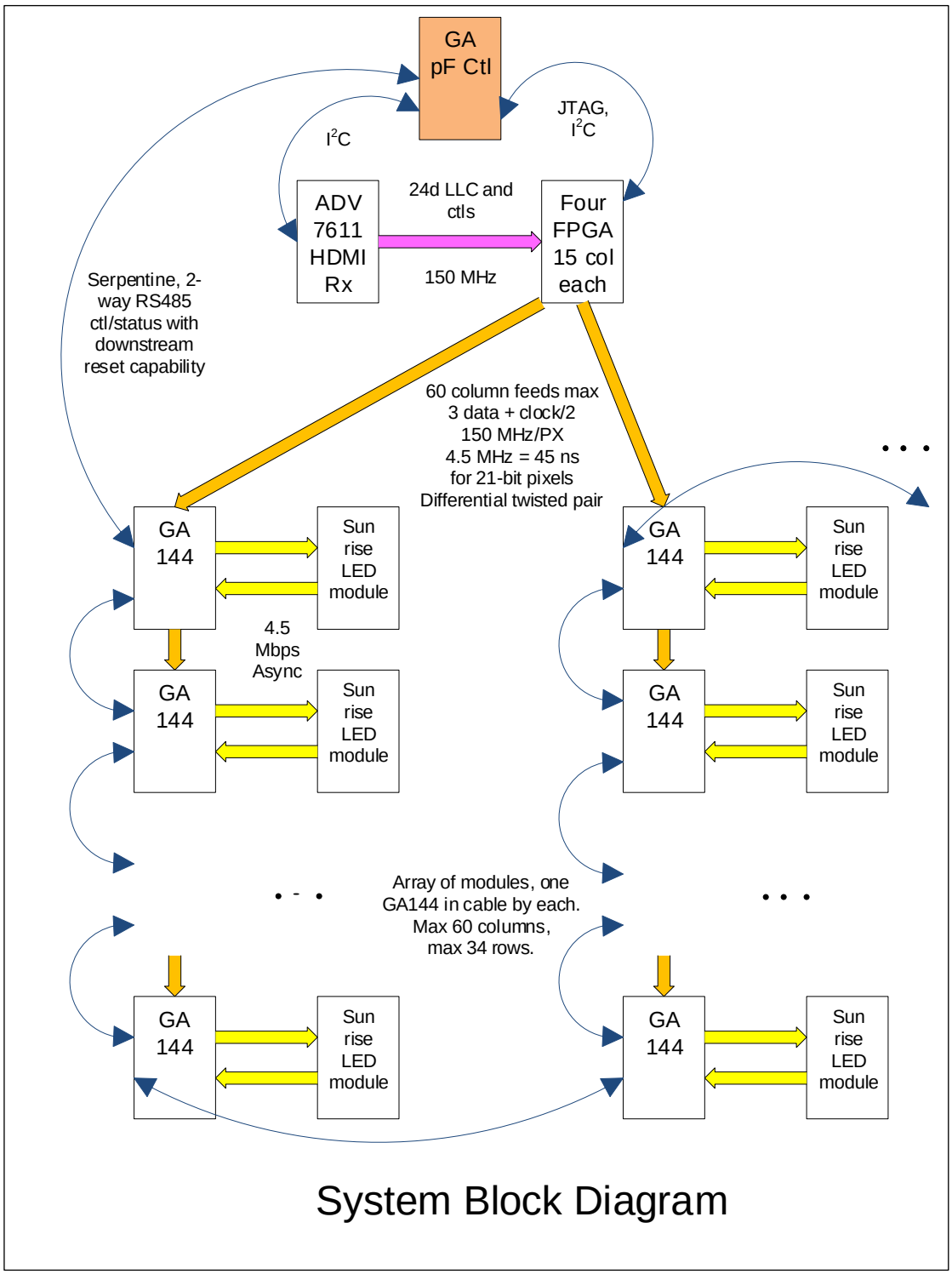


1.5.4 Next Steps

With a sustained pixel time of 319.68 ns down the column and worst timing (MX of 48), 2 data bits give us 26.640 ns per word for 24-bit pixels and 29.061 ns per word for 21-bit pixels. 3 data bits give us 39.960 ns for 24-bit pixels and 45.668 ns for 21-bit pixels.

2. System Design

Two PCBs are included: Control PCB, with HDMI video receiver, top level distribution, configuration and overall control/status duties; and Module PCB, one per Sunrise module, for video distribution down each column. This design requires some hardware difference between odd and even columns, either in the cabling or in PCB optioning, to implement the serpentine control channel. Five wire pairs go down each column: Four to distribute video and one for control/status channel.



System Block Diagram

2.1 System Control PCB

This board configures and controls the display as a whole. In this section we walk through the schematic for the board in the order of its hierarchy. Later on we will discuss the protocols implemented by software, and testing considerations.

The first page of the drawing is labeled "**Top**" and shows the highest level block diagram of the circuitry, showing the relationships between the **Computer** (two GA144s as a polyFORTH system), the **ADV7611** which receives 1080p video over an HDMI link, the **Video** section comprised of four FPGAs which decompose the 1080p signal into lower speed LVDS feeds for up to sixty columns of Sunrise LED modules, and the **Power** supply.

2.1.1 Power Section

The main supply, powering everything but the four FPGAs and sixty 4-bit LVDS line drivers, is adapted from Analog Devices' evaluation board and comprises the last three pages of the schematic. It produces 1.8V, 3.3V and 5V from an external wall-wart. This supply uses linear regulators and will probably be simplified for production units. 3-position jumpers allow for current measurement on the 8 supply sections and also allow selection of external power from a barrier strip. These sections are:

Name	Voltage	Est Current	Chip/section	Nomenclature
VGA_1V8	1.8	TBD	GA144s & related	
VGA_3V3	3.3	TBD	I2C, JTAG & related	
VGA_5V	5.0	<500mA	Ether transmit	
VAC_1V8	1.8	115mA	ADV7611 Comparator	
VAP_1V8	1.8	37mA	ADV7611 PLL	
VAD_1V8	1.8	189mA	ADV7611 Core	
VAT_3V3	3.3	58mA	ADV7611 Terminator	
VAIO_3V3	3.3	179mA	ADV7611 Outputs	

Because we won't know the power requirements for the FPGAs and the line drivers until the prototype is operating and we can take measurements, all power for these components is supplied externally using a barrier strip. Four supply sections each have 3 position jumpers for current measurement.

Name	Voltage	Est Current	Chip/section	Nomenclature
VXINT_1V8	1.8	TBD	FPGA Internal logic	
VXAUX_3V3	3.3	TBD	FPGA Aux pins	
VXIO_3V3	3.3	TBD	FPGA Outputs	
VCD_3V3	3.3	TBD	Column line drivers	

2.1.2 Computer Section

The second page of the schematic is the heart of this section. It consists of the polyFORTH system and internal system I/O, a page of Watchdog and reset circuitry, and a page of 10baseT Ethernet line interface circuitry.

2.1.2.1 polyFORTH system

We have borrowed circuitry from the GreenArrays EVB001 evaluation board, including the pair of GA144s, the 2 MB SRAM, both 1MB and 16MB SPI flash for boot, operating software and FPGA programming strings, two FTDI USB to serial interfaces, the ability for USB port A to reset chip 0 and the flash, and the signals for chip 0 to boot and control chip 1. This allows the EVB001 self-test code to run on the two GA144s, omitting the SERDES tests because none of them is connected. It also allows polyFORTH to run without change.

The 10baseT Ethernet cicuitry is included, however using a 10 MHz crystal instead of a Fox oscillator. A second SPI flash, this one with 16 MB of storage, is included and jumper block substitutes it for the conventional 1 MB part. Both of these features will require code changes; Phase jitter interpolation to initially excite the crystal, and a different writing mode for the large SPI flash (the AAI write function has been deprecated.)

I/O pin allocation is as follows:

Pin(s)	Functions	Remarks
705.1,3,5,7	SPI flash boot & disk	Standard pF usage.
008.1,3,5,7	SRAM control	
007.d0-17		
009.a0-17		
708.1,17	Async boot, IDE	Development and test/commission.
100.17, 200.17	Async pF terminal	Development and configuration.
500.17	Reset to Chip 1	Weak PD resets chip 1.
300.1,17	Sync boot & bridge to chip 1	Standard pF usage.
417.17, 517.17	Ethernet 10 MHz Crystal	
317.17	Ethernet Transmit	
217.17	Ethernet Receive	
600.17	INT1 from ADV7611	HDMI status changes from ADV7611
715.17	Unused pin	Brought out to test point
10300.1,17	Sync boot & bridge from chip 0	Standard pF usage.
10715.17	Xtal 9 MHz	For control channel baud rate
10708.1,17	Dir (Tx) & data for ctl channel	Pulled high on reset
10317.17	PGM- signal for FPGAs	Weak PD on reset
10217.17	INIT- signal for FPGAs	Weak PD on reset
10008.1,3,5,17	(JTAG) TDI, TCK, TMS, TDO	For programming FPGAs and also might be usable for later comms. Pulled high on reset.
10517,17	RESET- to ADV7611	Pulled high on reset.
10705.1,17	(I2C) SDA, SCL	For ADV7611 and, if needed, FPGAs. Pulled high on reset.
10200.17	Enable Watchdog	Weak PD on reset.
10417.17	Watchdog Retrigger	Weak PD on reset.
10100, 10500, 10600.17, 10709.ao	Unused pins	Brought out to test point

After booting, polyFORTH is responsible for configuring the FPGAs and the ADV7611 HDMI video source chip. The system is capable of fully automatic and/or interactive control. Chip 1 is used for I/O expansion and is controlled by the host using Mark 2 snorkel/ganglia over the synchronous port bridge.

Jumpers in the Computer section are as follow:

Pin(s)	Default	Alternate
J101	3-5: Enables 1 MB flash.	1-3: Disables 1 MB flash.
J101	2-4: Disables 16 MB flash.	4-6: Enables 16 MB flash.
J102	Open: Enables SPI flash boot.	1-2: No boot.
J106	Open: Disables USB reset.	1-2: Enables USB reset.
J203	2-3: Watchdog always enabled.	1-2: Watchdog enabled by software.

2.1.2.1.1 Lessons Learned from Prototype

Crystal footprint for Y101, still 3-pin, connects signal to center pin not end pin. Jumpered pins 2 and 3 to make this work, layout must be corrected. Also, 0Ω resistor R116 is not populated and must be shorted to allow the crystal to oscillate.

2.1.2.2 Reset circuits

For the GA144 system we have a BU4816 reset chip and an STWD100NYWY3F watchdog timer. These will be evaluated for reliability in the prototype.

2.1.2.3 Ethernet Electrical Interface

This circuit provides the electrical interface to allow three nodes with GPIO pins to connect to a 10baseT circuit. One pin runs the 10 MHz crystal which we are basically required to use as a transmit frequency reference. One acts as a transmitter using a dual op-amp to make the higher voltage differential swings required, and uses a standard transformer to interface with the Cat5 cable. The receive circuitry consists of the other half of this transformer and a series current limiting resistor to feed a GA144 pin.

2.1.2.3.1 Lessons Learned from Prototype

Crystal footprint for Y301, still 3-pin, connects signal to center pin not end pin. Jumpered pins 2 and 3 to make this work, layout must be corrected. Also, 0Ω resistor R301 is not populated and must be shorted to allow the crystal to oscillate.

Our crystal oscillators will stop if disturbed with a probe on the crystal pin. They will also not start if another GA144 pin is connected to them but not set to high impedance. Evidence that a crystal is running must be obtained from a monitoring node.

2.1.2.4 Interchip Communications

ADV7611 configuration is done as described in AN018.

Target node 10708 masters the 400 KHz (max) I^2C bus for the HDMI chip (10708.15 is **SCL**, 10708.1 is **SDA**). 10517.15 is **RESET-** for the HDMI chip. **INT1** from HDMI is received by Host pin 600.17. All four of these pins use a TI TXS0108 level shifter to translate between 1.8V and 3.3V in an open-drain configuration.

For the FPGAs, we reset the device by pulling **PROGRAM_B** and **INIT_B** low. Raise **Program_B** T_{PL} (4 ms) before raising **INIT_B** and raise **INIT_B** at least T_{POR} (40 ms) after starting the reset.

We talk to the FPGAs using six lines shifted to 3.3V by a TXS0108 open-drain level shifter. 10217.17 drives **INIT-** and 10317.17 controls **PROGRAM-** while node 10008 pins 17, 5, 3 and 1 connect with JTAG signals **TDO**, **TMS**, **TCK** and **TDI** respectively. In addition the FPGAs are part of the I^2C bus until we learn whether JTAG alone is sufficient for everything we need to do. The prime reason for using open-drain level shifting is to accommodate the JTAG connector that will be used for initial debugging of the FPGA configuration.

2.1.3 ADV7611 Video Receiver

We have borrowed circuitry recommended by Analog Devices for its ADV7611. This chip is configured, decoupled and powered as recommended by Analog Devices. We configure it to demand 1080p video at 60 Hz with fully standard sync and frame timing.

2.1.3.1 Problems with Analog Devices design

In using the evaluation board from Analog Devices the following problems were observed:

1. HDMI hot plug detection: There is basically no load on the +5V line from the HDMI source. As a result we do not perceive that the cable has been unplugged and the HDMI Hot Plug detect line is left high. We see the LED on the eval board is on, but that LED is powered by the onboard +5V and not by the cable. To alleviate this we have added a 5K resistor to ground from the **DDC_+5V** line.

2.1.3.2 Procedures

RESET- is asserted for 10 ms after which we wait 10 ms before talking to the chip. After reset, all internal registers are in their default states and the HDMI hot plug detect signal is driven low by the **HPA_A/INT2** pin. By default settings, this signal will be released to go high 1 second after reset if an HDMI source is plugged in.

When hot plug detect is low, sources are supposed to assume sink is not up and are not supposed to try talking to them. When the hot plug detect signal goes high, a valid EDID 1.3 as defined in CEA-861-D must be available along with a CEA EDID Timing Extension Version 3 that includes an HDMI Vendor Specific Data Block, all per 8.3 in the HDMI spec. We must therefore make our EDID ready to read within 1 second after resetting the chip, whether or not an HDMI cable is plugged in. If we later on change the EDID, we are required to signal the source by driving hot plug detect low for at least 100 ms.

Thus, immediately after ADV7611 reset our critical path actions are to initialize registers as recommended by ADI and then to make the EDID ready to be read. Having done those things within the required time we are free to set the rest of the chip up.

CEC Physical Address Discovery (HDMI 8.7.2) is mandatory even if we are not CEC-capable; we appear to comply by simply putting the default address into the EDID structure.

2.1.4 Video Section

This consists of four FPGAs on the pixel bus, with provision for daisy chained JTAG configuration, either by the GA144 or by a Xilinx programmer, and I^2C communications for managing them later. The I^2C addresses of the four FPGAs are A0, A2, A4 and A6; this is in order down the bus starting at the ADV7611, in order of the JTAG daisy chain, and in order left to right of the four 15-column segments of the LED display.

Each FPGA drives 15 columns of modules with 4-bit LVDS cables (we assume Cat5.) For each of these there is a quad LVDS line driver and an RJ45 connector. The FPGA specification is as follows:

2.1.4.1 Lessons Learned from Prototype

The pixel bus (24 bits plus VS, HS, DE and clock LLC) laid out in the way we planned (pixel bus straight across from the ADV7611 through the four FPGAs on the bottom of the board; all crossing signals are routed on the top and middle signal levels) is much cleaner than was the same bus on the Analog Devices evaluation board. With no power to any of the FPGAs, the signals to be sampled on the rising clock edge seem to be firmly in their desired states when the edge comes by, as seen here: Top trace is LLC, bottom is B7 with a transition from 0 to 255 on blue.

2.1.5 FPGA Specification

The problem statement for the FPGA is simple, but timing is critical. The substantial pin-out requirement in combination with Sunrise preference of FPGAs require that four FPGAs be used.

Input to the FPGA is a stream of pixels supplied by the ADV7611 HDMI chip as 24-bit parallel data clocked at 150 MHz by the LLC signal, and qualified by the framing control signals.

2.1.5.1 Framing Awareness

Firstly the FPGA must discriminate between the horizontal and vertical sync delays and the actual delivery of 1920x1080 pixels. No pixels will be processed during those delays; i.e. the FPGA will receive only 1920x1080 LLC events as incoming pixels per frame.

The FPGA must be configurable to isolate a rectangular window, corresponding with the physical display, from each incoming frame. The origin of this window is expressed as W0X and W0Y coordinates origined in the upper left corner, which may be zero, and the size as WNX and WNY such that WNX ≤ (1920-W0X) and WNY ≤ (1080-W0Y). WNX must be divisible by PX so that the window width is an integer number of module widths in pixels. Later on we added registers for origin and width of each column so that columns of differing widths could be used to build a sign.

The first W0Y lines of each image (if W0Y nonzero) will be ignored and the following WNY lines processed in each frame. For each processed line, the first W0X pixels (if W0X nonzero) will be ignored and the following WNX pixels processed in each frame.

2.1.5.2 Pixel Processing

Each of the first PX pixels in the active window of each scan line will be inserted into a FIFO for transmission to column zero modules. The next PX pixels will be inserted into a FIFO for transmission to column 1, and so on for MX FIFOs. We believe that the limit for MX is 60 in any practical configuration, so there will be 15 FIFOs in each FPGA (for maximum MX of 60) and each will have a depth of 40 pixels (the maximum PX).

The output side of each FIFO, as soon as it is not empty, will feed a serializer that transmits each pixel as seven (21-bit) or eight (24-bit) parallel words of three data bits, using straight DC level coding, and a half-frequency clock. Setup time is provided by the GA144s, so all four lines should be driven with no skew. These four signals must be 3.3V single ended CMOS (push-pull) logic.

The clock for outbound 3-bit words must be derived from LLC but will be divided down by a factor of MX (MX*2 given that it is a half frequency clock.) Note that this means each of the FIFOs will be sending data down its column as soon as the first pixel is in the FIFO, and throughout the remainder of the current scan line, the next horizontal blanking interval, and into the next scan line a variable distance depending on which column it is, so that the FIFO is emptied before the first pixel for that column in the next line is added to the FIFO. Because transmission down a column continues well into the next scan line, this would lead to complications if LLC is only driven during the active part of each frame; however, that is not a problem with the ADV7611 whose LLC is continuous.

2.1.5.3 Configuration Control

The key parameters that will vary by installation are W0X, W0Y, WNX, WNY and NX. Initially we thought these would be loaded via JTAG, but we later decided to use I^2C so parameter changes wouldn't require re-programming the FPGAs.

2.1.5.4 Possible Complications

Although we initially assumed that any display would be built as an array of identical modules, it soon became apparent that to meet customer requirements for particular sign sizes there was a good chance that different modules may be used on the bottom or right-hand edges, for example, so the FPGAs gained more configuration registers.

2.1.5.5 Clarifications to the FPGA Specification

The clock used for transmitting 3-bit values down a column from its FIFO will be derived from the pixel clock using a fixed integer divisor; if a single divisor value is needed it will be 6 (correct for 48 columns), giving 25 megatriads per second (40 ns period) and a square wave clock frequency of 12.5 MHz. When a FIFO has been emptied, the clock output will stop generating transitions but will remain in the same state (high or low) as it was after the edge clocking the last triad sent. The edge for clocking the first triad of the next line will begin at this state. There will be no discontinuity in this clock signal across frame boundaries and there will be no spurious edges sent; each edge sent will clock three bits of data that will be taken seriously.

The FIFOs will be filled, and emptied, only for the 1080 active rows of a 1080p display. Silent clock line is used by the VMODs to detect end and start of frame.

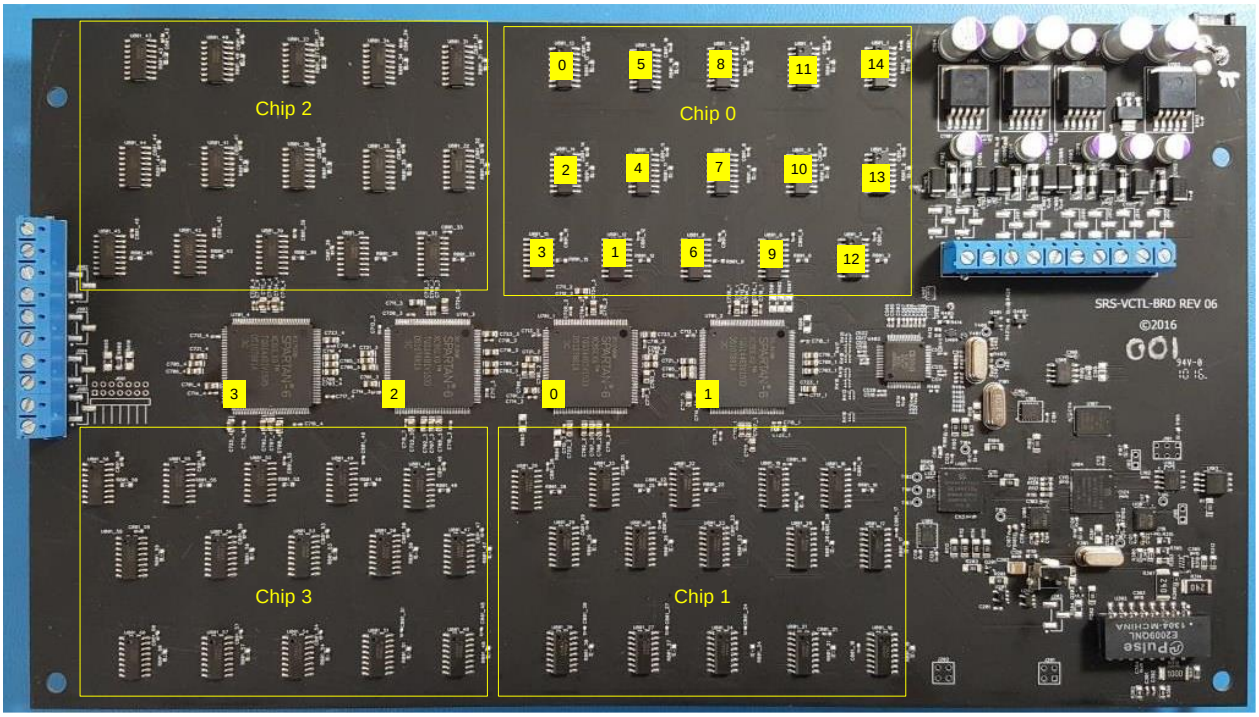
2.1.5.6 I²C Registers and Functions

The four FPGA chips 0..3 are addressed as I2C devices xA0, xA2, xA4, and xA6 respectively. Within each FPGA the registers are assigned as follow:

Address	Bytes	R/W	Default	Significance
x00	2	RO	x5501	ID code
x01	2	RO	(s/b 2200)	Current pixels per HSYNC line
x40+n	2	RW	40	Column ***n*** width in pixels [1..40] ***n*** in [0..15] (PX_n)
x80+n	2	RW	40n	Column ***n*** pixel origin [0..1919] ***n*** in [0..15] ($W0X_n$)
xC0	2	RW	0	Top row of sample window [0..1079] (W0Y)

2.1.6 Roadmap

The layout of FPGAs and associated RJs on the back of the board is not quite sequential. Here's the top side:



Top side of VCTL PCB – Each chip's columns same order, RJs on bottom same pattern as drivers on top

The sequence of FPGA numbers above reflects their sequence in the TDI-TDO daisy chain. Recommend no change in the planned SPI addressing for the FPGAs; the connector grouping is rational enough as it is and 0/1 were intended to feed the leftmost columns as the layout supports. We may wish to rename the pins on the FPGA so that the numbering of connectors in each group of 15 is more rational; one simple fix would be to renumber columns 0, 1, 2, 3 so that they fit the pattern of the remaining columns 4..14. Once this has been determined we can come up with a reasonable silkscreen for the backside of the PCB.

2.1.7 ADV7611 Observations

After power-up:

```
300 LOAD ok
+ADCHIP ok
.AD
Chip 2051  HDMI 5V NO.  Assert HPA NO.  TDMS clk detect NO.
  TDMS PLL locked NO.  VS Lock NO.  DE Lock NO.  HDMI PLL Lock NO.
  STDI laced NO.  CP proc lace NO.  Lace conflicts YES.
HDMI section mode DVI  HS,VS high 00 X=0 Y=0
  Bits/gun=8  Interlaced NO.  FIFO locked NO.
  Line wid 1  Front 0  HS 1  Back 0
  Fld hght 0  Front 1  VS 0  Back 8191
  TDMS 0 kHz.  Input colors 0 (0=RGB, 1=full)
CP section CSC BYPASS.  STDI valid NO.  Lines/frm=0
  Laced NO.  Xtal/8Lin=16383  Vsync=0  Xtal/fld=0
  CP Free-running w/mode&std.  CP Free-running: YES.
INT1 NO.  STDI valid NO.
io/42 3  CP Unlocked
io/47 6  Undefined
io/5B 2  CP input has changed from locked to unlocked
io/65 6  Audio is muted ok
```

After plugging into Win7 notebook bat and sending video (identify):

```
.AD
Chip 2051  HDMI 5V YES.  Assert HPA YES.  TDMS clk detect YES.
  TDMS PLL locked YES.  VS Lock YES.  DE Lock YES.  HDMI PLL Lock YES.
  STDI laced NO.  CP proc lace NO.  Lace conflicts YES.
HDMI section mode DVI  HS,VS high 11 X=1920 Y=1080
  Bits/gun=8  Interlaced NO.  FIFO locked YES.
  Line wid 2200  Front 88  HS 44  Back 148
  Fld hght 1125  Front 4  VS 5  Back 36
  TDMS 148484 kHz.  Input colors 0 (0=RGB, 1=full)
CP section CSC BYPASS.  STDI valid YES.  Lines/frm=1124
  Laced NO.  Xtal/8Lin=3385  Vsync=5  Xtal/fld=1864
  CP Free-run buff locked.  CP Free-running: NO.
INT1 NO.  STDI valid YES.
io/42 2  CP Locked
io/42 4  STDI Data Valid
io/47 6  Undefined
io/5B 1  STDI Data is valid for sync channel 1
io/5B 3  CP input has changed from unlocked to locked
io/65 6  Audio is muted
io/6A 0  DE Regeneration is locked to incoming DE signal
io/6A 1  VS Filter has locked and vertical sync params are valid
io/6A 4  TDMS Clock is detected
io/6A 6  TDMS PLL is locked to incoming clock
io/6F 0  HDMI cable +5V is detected
io/83 1  TDMS Freq has changed by more than FREQTOLERANCE
io/83 6  Irregular or missing pulses have been detected in TDMS clock ok
```

2.2 Production VCTL PCB

FTDI FT232R chips need to be configured. Do the normal sequence of actions in the PCBCOM document with differences: In place of step 5, change two string descriptors:

- Product Description is `Sunrise VCTL Port A (and B)`
- Serial Number is `GAVCTLnnnA (and B)` with first chip 001. ("Device Instance Path")

2.2.1 Notes for Production Board

- Add LEDs to show status. This board has no visible indications of its condition without hookup to serial or network.
- The new SPI flashes (without reset lines) require that we put a 1k pull-up resistor on GA144 zero 705.3 going to **CE-** line on the flash, so that we can boot again after GA144 reset.

2.3 VCTL Test & Commission

To manufacture the VCTL in production, procedures and jigs for testing and commissioning will be required.

2.4 Module PCB

Because this board will be produced in volume, we must minimize cost of manufacture. The prototyping board is necessarily larger and more costly than the production target because we must be able to experiment with changes in the critical parts, particularly line drive and receive, reset and watchdog circuits, as well as final bit assignment in parallel ports for optimal code. The flash is only used for booting and nonvolatile configuration data; its size may be reduced if it will save cost. In this section we walk through the prototype schematic.

Lessons Learned from Prototype:

The power consumption of this board, while small, is considerable when multiplied by 30 or so for the number of modules in a column. Power daisy chaining must be considered carefully as to board connection method.

The massive power spike when the board is first reset after power-up (on the order of 250 to 300 mA per board) is probably going to be intolerable. This spike is due to having all of node 007 unused parallel input pins connected together and grounded through a pull-down resistor. Until we are able to get to node 7, change outputs to all ground (for no spike next reset), and switch the node to input mode, the drivers are all shorted together with indeterminate values. The alternatives are to put 14 separate pull-downs on the unused pins (cleaner design, higher cost) or leave them open (which has the potential for bringing metastable voltages into the ALU) and spend additional time and resources masking the value off in the input path.

2.4.1 GA144 Page

This page covers the GA144, its decoupling caps, and SPI boot flash; in normal mode, the boot flash is enabled.

For prototype only, there are hole patterns (not populated with headers) serving as test points for the boot flash, and patterns for selecting which GPIO pins are used for other functions (crystal, serial to and from LED module, serial to and from neighbor modules for control/status/configuration interface). There's no pattern for crystal itself because that will just be soldered down to holes in the patterns J4 and j5 for now (actually plugged into header pins.)

To interactively program this board we'll use a SparkFun FTDI module plugged into the J102/J103 patterns, in conjunction with no-boot jumper J101 for flash and the FTDI reset enable jumper J108. We are suggesting that production testing (and, if necessary, flash programming) will be best done using a bed of nails that will contact exposed pads for IDE serial on the bottom side of the board, along with pads for the no-boot function, to minimize manufacturing cost of these boards.

Test points are provided for video clock+ out, and clock and bit 0 inputs, on the pixel pipeline at the GA144.

2.4.1.1 Lessons Learned from Prototype

Low voltage testing with IDE fails at about 1.62V; suspect it is the FTDI chip that fails us here. Will have to do low V testing after we have an array of boards running without assistance.

The newer SPI flash chips do not have a reset line as such. It appears we have to allow chip select to go high in order to reset that sort of chip. We will have to put a pull-up resistor on the chip select line on the production board.

The flash is active for about 18.4ms after reset is released, with only half the RAM in every node being loaded with ganglia. The actual time used will probably be on the order of twice that by the time the whole application is being loaded, and longer still with the lower speed flashes.

The time required for ameliorating the shorts on node 007 outputs, the first time the VMOD is powered up, is about 1ms after rise of the reset signal. This will probably be slower with the lower speed flashes. As it turned out it was moot because the parallel port shorts dragged supply voltage down too low for the flash to work at all.

2.4.2 Power Page

Incoming 1.8 and 3.3V supplies are decoupled and provision is made for power measurement. All six headers on this section should be populated with male pins. Test points are provided for supply buses.

2.4.2.1 Lessons Learned from Prototype

We screwed up with the power inputs. Rather than two separate 1x3 headers we should have used a single 2x5, reducing chances of error and increasing sturdiness of connection.

The production boards are going to need a very good way of connecting their power.

Power Budget

An idle chip, not driving differential video output but with parallel port set correctly, burns about 5 mA. When the video bus drivers are turned on (driving differentially), the terminations burn another 25 to 28 mA.

2.4.3 Pixel Pipeline Page

Eight lines from pins on node 009 (Port8) drive the cat5 cable through the resistor network Warren designed. An RJ45 connector is used for this connection although in production it may be best to omit the connectors. The incoming video is connected by an RJ48 to another of Warren's networks feeding an LVDS receiver which is then connected to four pins of node 007 and pin 008.17 (for clock only) of the GA144 using voltage dividers to step down from 3.3V to 1.8V. In production this might be simplified to series current limiting resistors only. Provision is made for caps in case there's incoming noise to suppress, and for differential probing of data line 0 at the input to the LVDS receiver chip. Preliminary work with these networks and 2 feet of Cat5e cable indicates there will be about 700 mV peak to peak on the incoming differential receiver pins with good, crisp rail to rail signals and only a little ringing.

The four signals on the pixel daisy chain, with their cable pin-out and GA144 connections, are as follow:

Signal	GA144 Pins		RJ45 pins		GA144 Rx pin	Second Rx pin
	Tx+	Tx-	+	-		
Clk/2	A10	A11	1	2	D03	
Bit 2	A12	A13	3	6	D00	
Bit 1	A14	A15	4	5	D01	
Bit 0	A16	A17	7	8	D02	008.17

Having determined that the GA144 will need to do table lookup both for interpreting incoming data and for generating outgoing differential signals, the above arbitrary bit assignments have no effect on code design.

2.4.3.1 Lessons Learned from Prototype

Each transmit pair uses negligible current when both pins driven same, but on the order of 6 mA when driven differentially due to the load resistors. It looks like 25-28mA per module for this alone.

The 10kΩ voltage divider stepping the 3.3V outputs of the receiver chips down to 1.8V was a mistake, rolling off the incoming signals too far to be useful. Rescaling it to a 1k divider gives much more usable signals.

The incoming signal on D0 with 90pF parallel cap is considerably cleaned up relative to the other three lines. It may be worth treating the other three lines the same way.

A common mode observation: At one point the VCTL was powered on an old ball and tube ungrounded 110VAC circuit that shares its common with a second circuit, while the VMODs were powered by a new circuit that had a solid ground. The measured voltage between ground on the second circuit and the ground lug of the power supplies for the VCTL was on the order of 12VDC and 67VAC. Video was nevertheless working, most of the time, although periodically a horizontal band of grizzling noise would appear and "roll" down two of the columns. This would seem to imply pretty good common mode rejection on the quad line receivers we chose!

2.4.4 RS-485/422 (4)

This sheet has RS485 transceivers for the control/status daisy chain (max 250 kbit/s), RS422 transmitter and receiver for talking with the Sunrise LED module, and level shifters for all those. 3.3V transceivers are used because this gives better common mode capability; those with 3.3V differential sides and 1.8V logic sides are more expensive than is the combination of 3.3V-only parts with a level shifter. RJ45s are used for these interfaces although in production these connectors will probably be omitted.

2.4.4.1 Control/Status Daisy Chain

Note! The simplest/cheapest way we see to manage the control channel is a single serpentine daisy chain fed by a single RS485 interface on the Control Board, leading to the top of the left most column (column zero) of modules. The daisy chain proceeds, with half duplex single-pair RS485 connections, down that column with each module's "next" line connected to the "previous" input of the next lower module. From the bottom most module of column zero, its "next" line connects to the "previous" line of the bottom module in column 1. The daisy chain proceeds upward from there with each module's "next" line connected to the "previous" line of the next upper module. At the top of column 1, that module's "next" feeds the "previous" line of the top module in column 2, and so on.

This requires wiring odd and even numbered column distribution assemblies differently. There might be other ways but that would imply different code in modules destined for odd or even columns, with different floor planning problems, or else jumpering or other trace routing options to configure a board for odd or even columns. We think the wiring is the lesser of evils, but the trace routing option would work equally well without software change... although once assembled the odd and even cable assemblies would still not be interchangeable, and if the trace routing option was destructive the PCBs themselves would not be, either.

With the serpentine method, reset can be propagated through the daisy chain for sign power-up, and it is possible to plan for selective reset of a given module. Sequential serpentine module addresses become known to each module during power-up and can be stored in flash. Once all modules have unique addresses, each can be informed of its configuration parameters including position in column, dimensions of associated LED module, and so on, all of which is known to the control processor.

This page includes a serpentine reset circuit that allows the previous module in the control/status daisy chain to reset us by holding its signal in the physical low state (same state as start bit but for a very long time like a BREAK signal). *When this line is in input mode* it is monitored for a reset signal. RC network R406 (10k) and C413 (1 uf) accumulates charge with a 10 ms time constant. Whenever the signal line from previous module goes physical high (stop bit, idle), this charge is shorted to ground through an N transistor. When the charge is allowed to build to normal V_{IH} for 1.8V CMOS parts, something on the order of 7 ms, a low signal is sent to the main reset circuit. We will probably be running at least on the order of 9600 baud so this should be safe. The time constants of all modules are additive for resetting the whole sign; 2000 times 7 ms is already 14 seconds, so this could get out of hand in a hurry.

Test points are provided for differential probing of the data lines from the previous module PCB. Jumper J403 provides for disabling the daisy chain reset.

The RS485 transceiver circuitry is presently wired such that rest state is high on the single-ended lines, consistent with the fail-safe mode of the receivers. TR+ (pin 2) of the connectors is high and TR- low in rest state. On the module boards, the data lines for all outputs are pulled weakly high (10K) to overcome the weak pull-downs (40K) when the GA144 is in reset; and the direction lines for the transceivers are weakly pulled down to ensure input mode on both sides of the daisy chain. Data lines on the control board are pulled up as well. Thus the states of all the lines are high when the GA144 is in reset, when the lines are idle, and when a cable is open or unplugged. Note that this is the opposite polarity as normally used for GA144 serial I/O.

The RS485 line from the previous module is in input mode at reset and will be in input mode at all other times except when we are transmitting to the previous module. Thus, normally we can be reset by the preceding module. *Warning: If a module ever sticks in output mode on its line to the preceding module, the stuck module may no longer be reset by the daisy chain until it becomes un-stuck or power is cycled.*

2.4.4.1.1 Lessons Learned from Prototype

The ISL83483 does not seem to have a normal high failsafe on open input, at least not reliably; same type of behavior as noted below for the ARM interface. This means that loss of input connection could put us in reset.

It also means that we cannot depend on the fail-safe mode to prevent the next module from resetting while the current module does. Pull-ups should be added to both the data and direction lines (Port10.3 and 4 respectively) and the pull-down on Port10.4 removed so that we drive control to the next module high while in reset.

Observed rise time of C413 is about 7.45ms ground to 0.9V and the time to actual RSTin assertion is about 7.85ms. These timings are consistent with the 10ms TC of the circuit. Fall time from 1.8V to ground is about 5µs so as long as a bit time is greater than that, and as long as character frame time is considerably less than 7.5ms, this set of values will work well. ***For a faster serpentine reset, run at a high bit rate, shorten the time constant and use precision R406 and C413 to minimize TC variability.***

2.4.4.2 RS485 4-wire ARM Interface

Communications to and from the ARM on the Sunrise module use two wire pairs, connecting to J6 on the Sunrise module. Pins 1 and 2 are **TX-R-** and **TX_R+** respectively, signals from the ARM. Pins 3 and 6 are **RX_R+** and **RX_R-** respectively, signals to the ARM. Polarities are as above, with the single logic lines high, and the + side of a pair higher than -, when the line is at rest (stop bit state). A weak pull-up resistor is used to maintain the single ended output line high when the GA144 is reset. Test points are provided for differential probing of the data coming from the ARM.

2.4.4.2.1 Lessons Learned from Prototype

The ISL83491 does not seem to have a normal high-failsafe on open inputs even though the data sheet says it does! Measuring 0.228V on pin 12 and 0.227V on pin 11 I see pin 2 , the RX pin, is low! Boards 002 and 003 both behave this way. Added a 1k pull-up to pin 2 and still it is 0.03V so it's sure enough being driven.

2.4.5 WatchDog TImer (5)

The main reset circuit is also on this sheet. Reset is triggered by power-up, by a reset button on the prototype board only, by the reset signal received from the upstream control/status module as above, or, during development, by the FTDI reset signal; this is done using a commercial reset IC. In addition there is a watchdog circuit in this section, using a commercial chip set at nominally 1.6 seconds.

The reset circuit pulls **RST-** to ground as soon as its input signal **RSTin** is high enough to turn its output N transistor on, at about 0.70V. It releases **RST-** which rises with a TC of 110µs when the **RSTin** signal rises above 1.6V.

RSTin may be grounded by a reset button, by a signal from the FTDI serial dongle (if enabled by J108), by the watchdog circuit (if enabled by J501), or by BREAK on the incoming control channel (if enabled by J403). Each of these reset sources pulls **RSTin** down very quickly, and holds it down for a varying amount of time. The reset circuit does not assert **RST-** low while **RSTin** is at ground. It only does so when the **RSTin** line is released, at which time it rises with a TC of 10.1ms, producing assertion of **RST-** for about 12.5ms. This should be long enough to allow an SPI flash to complete an erase operation; if not, it can be extended to about 25.4ms by removing R409 from the circuit. *However, for reasons not yet understood, the actual duration of the **RST-** signal is only about 2.6ms.*

Jumper J502 2-3 sets normal mode in which the watchdog circuit is enabled whenever **RST-** is high; 1-2 allows software to control it, but we do not plan to test this mode unless required, because it weakens the whole concept. J501 is inserted to empower the watchdog circuit to reset the PCB, and it will do so unless there is a rising or falling edge on the **WDIN** net within 1.6s [1.12..2.24s] of end reset, or since the last such edge, at which it asserts **RSTin** low for 210ms [140..280ms]. Empirically, a chip that has not been programmed will, with the default jumpers, reset itself roughly every 1.75s. During testing, either straight IDE or with serial boot stream, J502 must be moved from 2-3 (normal) to 1-2 (disable).

2.4.5.1 Commissioning a prototype VMOD board:

1. Dismantle switch on Sparkfun FTDI dongle so that it will use the **VCCIO** we provide, and solder two 9-pin female headers to its underside. Follow procedures in PCBCOM document except that Product Description is VMOD `Serial Dongle` and Serial Number is GAVMODBOBnn with first chip 01.
2. Solder 10k resistor between **VGAI_1V8** and the **RST-** net, required by change to open drain reset chip.
3. Solder all missing headers to board except J503 and J504. Add ground rail between J106 and J107.
4. Add default jumper blocks **before testing**:

Block	Pins	Effect	Comment
J201	2-3	Enable 1.8V to Core	Center pins
J202	2-3	Enable 1.8V to I/O	Center pins
J203	2-3	Enable 3.3V to I/O	Center pins
J108	IN	Enable **RSTin** from FTDI	Note shortens reset pulse 10x
J502	1-2	Disable Watchdog	Toward center of board
J501	IN	Enable Watchdog Reset	Underneath board
J101	IN	No Boot	
J403	1-2	Disable Control Reset	Away from ground headers

5. **After testing** install the following:

GA144 Pin	J105	J104	Signal	Comment
600.17	1	2	PREV Tx Enable	
500.17	2	1	PREV Data	
300.17	3		XTAL monitor net	Wire Wrap to J105.7
300.1	4	15	WDIN Watchdog Toggle	3" (Yellow) jumper
200.17	5	3	NEXT Data	
100.17	6	4	NEXT Tx Enable	
217.17	7		XTAL monitor net	Wire Wrap to J105.3,10
317.17	8	6	Data out to ARM	
417.17	9	5	Data in from ARM	
517.17	10		XTAL monitor net	Wire Wrap from J105.7
708.17	11	8	FTDI Data In	
708.1	12	9	FTDI Data Out	
715.17	13	14	XTAL	Mounted on 2x2 hdr
<EWD>	20	16	EWD no GA144 control	Wire Wrap always disabled

6. Change jumpers for normal operation from boot flash:

Block	Pins	Effect	Comment
J502	2-3	Disable Watchdog	Toward edge of board
J101	OUT	Flash Boot	
J403	2-3	Enable Control Reset	Toward ground headers; do NOT do this if there's nothing driving our RS485 IN socket.

If you insert J101 to disable flash boot, you must also move J502 back to pins 1-2 because nothing will be keeping the watchdog happy.

2.4.5.2 Development Procedures:

Disable Watchdog to use straight IDE or serial stream boot, by moving J502 jumper from 1-2 (toward center of board) to 2-3 (toward edge of board). Once serial stream boot is finished you can move the jumper back, but doing so can lead to confusion so it is not recommended:

Don't enable watchdog when booting from serial stream; if the watchdog trips this reset won't reboot if the No Boot jumper is still in.

Insert "No Boot" jumper J101 when using straight IDE to generally access the chip. This will require disabling watchdog as noted above.

Note that IDE talk does a reset. To use the IDE in some combinations such as with flash enabled it may be necessary to say reset after talk before anything will work.

2.5 Production VMOD PCB

Changes for production began with the file SS-MOD01r6.dch received from Sunrise and dated 1/07/16 16:44. We have renamed this file SS-MOD01r7.dch because we actually delivered one in December called r6. This section documents each change we made to this r7 file in generating r8.

Note to Sunrise: We are not using the same DipTrace libraries Matt did. If a BOM change required changing a library part we have done so and the need to maintain Matt's library is indicated along with the change. Our libraries are part of the delivered package but there's no way we know of to reconcile ours with Matt's other than making the parallel changes manually on his.

In updating the BOM we have used a "code": Resistors and caps shown as 10% can be whatever precisions are handy for you, whereas those shown with tighter tolerances have values used to set time constants, signal amplitudes or proper terminations and should be as close to what we write as practical. We have reduced all discrete SMT components to the smallest footprints allowed by the values or power dissipations as appropriate. We have assumed that 0402 is the smallest you'd like to be picking and placing; if you actually want to mess with 0201 some but not all of the parts can be that small.

Please note that in producing BOM ordering by name and value, DipTrace combines tolerances. It's necessary to unravel the BOM by part to see the tolerances. This may or may not matter depending on how you choose to treat tolerances in manufacture.

We have not gone to the length of selecting smaller packages for the ICs on this PCB. If you wish to adopt smaller package sizes that is fine with us although given no test points it would be nice to stick with parts it's feasible to probe. Leadless packages have obvious disadvantages if anything needs to be measured or investigated.

The first set of changes reflects differences between r7 and the prototype boards as built:

1. Change U502 from BU4916 (reset chip with push-pull outputs) to BU4816 (open drain). The latter are much more readily available than are the former. *Library change (define BU4816).*
2. Change U101 from SST25WF080 to 080B, validated on several of the prototype VMOD boards. *Library change (define 080B and its part number.)*
3. Note that all the 74LVC1T45W6 parts were replaced on prototype with TI SN74LVC1T45DBVR due to availability issues. This substitution seemed to work fine. We have not changed the schematic or BOM to reflect this.

The second set pertain to changes we have made to the prototype boards as built to yield a working sign:

4. Add pull-up resistor R1 on VOUT of U502, 10k 1% 0402.
5. Change R309, R316, R327, R331 from 5.11k to 453Ω 1% 0603, to correct divider (3v across this).
6. Change R314, R319, R332, R333 from 5.62k to 562Ω 1% 0402, to correct divider (3v across this).

The third set pertains to conclusions drawn during test and debug:

7. Delete R103 and disconnect D04..D17 from it and from each other.
8. Add R2..R15 each 10k 10% 0402 pulling each unused input pin D04..D17 to ground. *Note we marked these as 10% in DipTrace but they come out as 1% on the BOM.*
9. Remove reset line to pin 7 of U101 (new chips have no reset.)
10. Tie pin 7 of U101 high (is now HOLD- instead of RST-).
11. Remove J101 NO-BOOT jumper and R101 conditional pull-up. Replace with T1, NO-BOOT from bed of nails.

12. Add R16 5.1K 10% 0402 pull-up on CE- line for U101 to force de-assertion of CE- during VMOD board reset.
13. Set C302, C304, C324, C312 as 90pF 5% 0402 on triad video in. These have a small effect on signal quality but may be necessary for noise in a full display and can be depopulated later in production of they prove unnecessary.
14. Remove C305, C308, C310, C313 as unnecessary.
15. Remove C403, C408, C416 as unnecessary.
16. Set R406 at 1% and C413 at 5% to control time constant in PREV reset circuit.
17. Set R504 at 1% and C106, C502, C504, C505 at 5% to control time constant in RSTin- circuit.
18. Remove R408 pulling NEXT transmit control down. Add R17 and R18 10K 10% 0402 to pull NEXT transmit and data lines high during reset, to avoid resetting the next board in the serpentine.
19. Change R505 to 5.1K 10% 0402 and add T2 DISABLEDOG- to disable watchdog from bed of nails. Remove C503 as unnecessary. Put two of the unused inverters U409.3 and U409.4 to drive R505 from RST- to isolate those signals.
20. Remove J102, J103 and J108. Replace with T4 IDE-IN, T5 IDE-OUT and T6 IDE-RST-
21. Remove J104 and J105. Make connections as follow:
    a. U102 pins 14(300.17), 51(217.17), 60(517.17) for 9MHz clock distribution.
    b. PREV-D (Port10.1) to U102 7(500.17) for PREV data
    c. PREV-TX (Port10.2) to U102.6(600.17) for PREV Transmit enable
    d. NEXT-D (Port10.3) to U102.18(200.17) for NEXT data
    e. NEXT-TX S(Port10.4) to U102.20(100.17) for NEXT Transmit enable
    f. ARM-IN(Port10.5) to U102.59(417.17) for Receive data from ARM
    g. ARM-OUT(Port10.6) to U102.52(317.17) for Transmit data to ARM
    h. T4 IDE-IN to U102.78(708.17) for Receive data from bed of nails for IDE operations
    i. T5 IDE-OUT to U102.79(708.1) for Transmit data to bed of nails for IDE operations
    j. WDIN (Port5.1) to U102.15(300.1) for toggle signal to watchdog chip.
    k. Add Y1 9MHz and connect between U102.71(715.17) and ground. *We did not update library for this crystal pattern. It has the same problem as did those on the VCTL board: The crystal is a 2-pin device while the pattern is 3 pins. The actual connections should be on pins 1 and 3 of the pattern; pattern pin 2 is N/C. There is probably a better pattern for this device but we don't know what it is. If you would rather use an SMT crystal that is no problem but let's put one of your selected parts on one of the prototype VMODs to make sure we can run it.*

The fourth set pertains to changes that may be made now that we are satisfied with the design:

22. Delete T106, 107, 108 on triad video input.
23. Delete T104, 105 on triad video output.
24. Delete T101, 102, 103 on SPI DI, DO, SCK lines.
25. Delete T201, T202, T203, T204, T205, T206 in power section.

26. Delete J201, J202, J203 for power measurements.
27. Replace J204, J205 with two pin headers for illustration.
28. Remove T301, T302, T303, T304 on triad video input.
29. Remove T401, T402, T403, T404 on PREV differential line; remove T405, T406, T407, T408 on ARM TX line.
30. Remove J403 and connect port10.8 directly to Q402 pin 3 so that PREV reset is always enabled. Delete R409 as it is not used in the prototype when operating normally.
31. Remove EWD signal from PORT5 to J502.
32. Remove J501 (Enable watchdog to reset). Remove J502 Soft-Enable and connect the signals that were on 3(RST-) and 2(Enable Watchdog circuit).
33. Remove J503 and J504. Render U409.5 and .6 inert.
34. Remove K501 reset button.
35. Remove J106, J107. Add T3 for ground.
36. Remove R410 and C417, not necessary.

The fifth set goes over the BOM, reducing discrete part sizes and adjusting tolerances to reflect real needs (Note, as above, that DipTrace seems to ignore differences in tolerance when merging parts of like value and package. Beware!)

37. Set R314, R319, R332, R333 as 0402 (the 453Ω resistors can lie between 3.3V and ground through 20Ω pin drivers until the chip is reset, while the 562Ω resistors only lie between 1.8V and ground through pin drivers; accordingly less power to dissipate.)
38. R102 can be 1K 10% 0402; all the GA144 pins it protects reset as input.
39. R402, 403, 411 are changed to 1210 because they terminate true RS485 transmitters.
40. R301, 302, 305, 306, 307, 310, 311, 313, 315, 318, 320, 321, 325, 326, 328, 330 are all set to 100 1% 0402.
41. R303, 304, 308, 312, 317, 322, 323, 329 are all set to 200 1% 0402.
42. R324 is set at 1K 10% 0402.
43. R401, 404, 405, 407, 501, 502 are set at 10K 10% 0402. Note that these are still shown in the BOM as 1%.
44. Reduce R406, R504 to 10K 1% 0402
45. Reduce R412 to 1K 10% 0402
46. Reduce R503 to 20K 10% 0402
47. Reduce all capacitors to 0402 in accordance with Class 2 X7R-MLCC voltage/value limits except the following which need larger packages in that series: C201, 203, 205 10µF 10% 0805.

2.6 VMOD Test & Commission

Initially this will be done using a PC running arrayFORTH talking to an FTDI chip that's part of the bed of nails. The FTDI Break-out board ought to be almost sufficient; it has at least four pins that we should be able to use as control outputs although we may have to add one or two transistors as noted below for the bed-of-nails. Three of these outputs are needed for reset, no-boot and disabling watchdog. The fourth could be used to turn power on and off. Some design is called for before this can be called complete. Meanwhile here are relevant notes:

2.6.1 Notes for Bed of Nails test/program facility

- This can be an FTDI Break-out board with a small amount of additional circuitry:
    - Connection to IO power and ground
    - Transmit and receive lines can go directly to the T4/T5 testpoints (check transmit/receive nomenclature and make it same as was done on the VMOD prototype), as can the RTS reset line to T6.
    - We need to control NO-BOOT at T1. This must be asserted (to prevent flash boot) with a controllable current source; so long as FTDI is being used we need to pick a pin and use it to control a P transistor so that when not sourcing (boot allowed and SO not encumbered) it is high impedance.
    - We need to control DISABLEDOG- at T2. This must be asserted (watchdog disabled) with a controllable current sink; so long as FTDI is being used we need to pick a pin and use it to control an N transistor so that when not sinking (watchdog allowed to reset the chip) it is high impedance.
- The chip itself can be given all internal self-test using the IDE capability.
- To test the I/O it is necessary to include turn-around connections as part of the bed of nails, preferably at the connection points rather than anywhere nearer to the GA144. Video out to video in all four differential pairs; PREV to NEXT one differential pair; ARM transmit to ARM receive one differential pair. Software will need to be written to test this stuff.
- We have specifically NOT shown the bed-of-nails test points for powering the board or for turning the communications around. We've only shown T1, T2, T4, T5 and T6 to be specific about where those signals will need to connect, and for the bed of nails these may well not be plated thru holes. (T2 is an exception; it's not necessarily the bed of nails grounding point but is provided as a probe ground and should be implemented as a plated thru hole so a pin can be soldered there. If Sunrise has a better way of doing this, great.)

2.7 Sign Commissioning & Maintenance

We will need to create diagnostic procedures for use in the field after a full sign has been assembled.

3. Software Walk-Through

3.1 Protocols

Three protocols are used, for moving pixels, for sending frame updates and configuration info to the ARM controlling each module, and for sign management.

3.1.1 Column Pixel Pipeline

Four UTP lines carry 3-bit data with a half-frequency clock (data on each clock edge) down each module column. Eight of these triads comprise a 24-bit pixel with a nominal triad interval of 40ns and pixel interval of 320ns. The pixels sent down a column are only those relevant to the modules in that column and are thus **PX** pixels from each of the scan lines covering the column. There are no markers other than a substantial time with no clock, indicating vertical sync. Thereafter the pipeline begins with the **PX** pixels destined for the top LED row of the top module, left to right, followed by the next line in that module, and so on. Because noise is possible if not unavoidable and because a windowed display will have its bottom rows truncated, the VMOD must respond appropriately if there are either fewer or more pixels (or triads) in a given frame than expected.

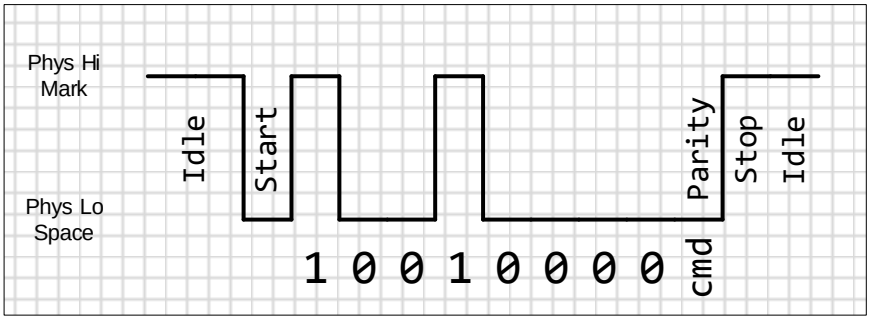
3.1.2 Sunrise ARM Protocol

The message format has been simplified and now does only what is necessary to deliver 24-bit pixels to the ARM and to configure the ARM for one of its four possible rotations. Accordingly characters are framed as 8 bits plus parity and one stop bit, with parity bit low for commands and high for data.

At present we define only one message, that which delivers a frame of pixel data to the ARM. This message does not cause a reply from the ARM.

3.1.2.1 Character Framing

The bytes we send look like this, where physical high state on single ended lines (and higher state on the + side of a differential line) is the rest state of the signal:



Sunrise ARM Command Character x09

We send command characters with space (0) parity and data with mark (1) parity. When the ARM is receiving our pixel data, expecting Mark (1) parity, any incoming character with a parity error is interpreted as being a command character and incoming pixel data processing stops immediately.

3.1.2.2 Data Message

A command character xC0, C1, xC2, or xC3 precedes 3840 characters of pixel data with each pixel represented by three consecutive characters giving values for Red, Green and Blue in that order.

If the ARM receives fewer than its module's size in pixels before the next command character, the missing pixels are simply not updated. If the ARM receives more than is module's size, those data are silently discarded. The first pixel following a command is stored in the upper left corner of the module and subsequent pixels are stored left to right in the top row, then left to right in the next row below it, and so on.

3.1.2.3 Commands

The low order three bits of commands indicate the orientation of the module, with zero being the default orientation of RJ connector openings DOWN as seen from front of module, and with 1, 2 and 3 indicating rotations of 90, 180 and 270 degrees counter clockwise from that default orientation (connector openings right, up, left respectively). This information is transmitted in every frame and the module ignores it unless the orientation in the command differs from that currently implemented in the module's tables. If there is a difference, the ARM ignores all subsequent commands and data until the table has been re-computed according to the new orientation.

3.1.3 Sign Management Protocol

An RS485 half-duplex control/status line originates at the VCTL and propagates through the array of VMODs in a serpentine fashion, starting with the module at the upper left corner of the display, proceeding down the leftmost column, then up the next column, and so on. Each VMOD has a line from the PREVious member of the chain and a separate line to the NEXT member.

In their idle, resting states, each member has its PREV connection in receive mode and its NEXT connection in transmit. The rest states of all the RS485 data lines are high.

When a PREV connection is in receive mode, it may receive commands and it may also be reset if the previous member of the chain sends a BREAK (holds line low for a defined period, currently specified as 10 ms.)

Messages are transmitted as sequences of 16-bit asynchronous characters (words) with two stop bits. Thus a character frame consists of 19 bits. The transmission frequency is 200 kbits/sec. These decisions were made for our convenience and have the side effect of increasing the workload and sophistication required of those who wish to reverse engineer components of our design and produce knock-offs of Sunrise hardware.

All messages originate at the initiative of the VCTL by transmission of one or more words to the first VMOD in the daisy chain. No VMOD transmits through its NEXT port, or back through its PREV port, except as directed by a particular command. All receive operations by everyone are subject to timeout, and the goal of timeout is that each VMOD will return to its idle state in a reasonable amount of time so that it may receive subsequent commands or resets from its PREV port. When a VMOD receives a message it does not understand, or times out during receipt, the VMOD silently returns to idle state.

The first word of each command identifies its function and, if unicast, the addressee to whom it is directed. This word is encoded as follows:

U	F	Address													
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

U denotes a unicast message intended for the module whose address is given. Else broadcast.
Function is decoded as shown below; (U) indicates unicast, (B) indicates broadcast.
Address is the sequential position of the addressed module in the serpentine. Zero does not exist. One is the module to which the VCTL board is directly connected. The field is zero if not used.

Because a word of all zero is what the receiver sees at the end of a BREAK, any command word of zero is ignored when received by a VMOD.

Every command is an exchange in which one or more words of feedback are expected. The feedback ends with a word of xFFFF. If that is the only feedback received, it is interpreted as a negative acknowledgment. (A default positive acknowledgment consists of a word of 0000 followed by a word of xFFFF.)

The large number of potential hops (>2000) and the long latency (100µs per hop) for a single word message to reach the far end of the daisy chain creates special problems for robust management of half duplex line management. One would ordinarily do this with time-outs, however that would require meticulous orchestration of a declining timeout at each step of the daisy chain to ensure that receivers near VCTL did not time out before those farther away did. We chose to simplify this entire issue by arranging that data are continually being sent back toward VCTL as messages propagate outward on the daisy chain. For one-word messages the back-channel is active almost continuously, while for the four-word configure message there is one word sent back for each four words transmitted. Every VMOD uses a receive time-out of 600µs on each word received, and for that matter the VCTL could do the same. When a VMOD at the end of the functional chain fails to reply within this interval, it really does not matter which NEXT receiver first notices the time-out, so long as the VCTL waits at least 600µs additional after receiving the xFFFF or timing out itself. From the VCTL's perspective, continuous incoming data are expected until someone detects a time-out, and that time-out event cannot follow the last valid reply in the stream by more than 600µs. Only a minimal wait is needed to allow the daisy chain to revert to outbound mode throughout its length under these conditions. By default this inbound traffic is generated by having each VMOD reply with its own address value as a form of acknowledgment that it's been forwarded.

The following messages are defined:

3.1.3.1 Enumerate (B) 00

The purpose of this message is to make each VMOD aware of its ***address*** in the daisy chain. The address of a VMOD is its *one-relative* sequential position in the daisy chain; it only needs to change if the geometry of the sign is reconfigured or if a VMOD is changed (replaced or moved.) The address is used in subsequent messages to direct an interaction with a specific VMOD. Upon receipt of **Enumerate** each VMOD takes the following steps:

1. Stores the address field from the message as its **address** in RAM for immediate use and starts in parallel an update of SPI flash variables to remember this address across resets and power cycles (this takes a long time, 75 to 100 ms, but the time is occupied in SPI nodes and does not affect control channel traffic.)
2. Sends a reply (the incoming sequence number) out the PREV port as a single word, acknowledging receipt.
3. Increments the sequence number and transmits **Enumerate** on the NEXT port.
4. Performs **PUMP:** Listens on the NEXT port, passing replies until timeout of 600µs (replies xFFFF) or receiving an xFFFF which is passed. Returns to idle after transmitting xFFFF out the PREV port for either reason.

3.1.3.2 Muster (B) 01

Each module sends back its four ***running*** variables (orientation, ypos, framesize, address) then passes the 1 word command along to NEXT and performs **PUMP.**

3.1.3.3 Reset (U) 10

If address does not match ours, acknowledges with own address, passes 1 word along to NEXT and performs **PUMP**.

Otherwise, Sends an acknowledgment (0000,xFFFF) out the PREV port and sends a RESET to the NEXT port.

3.1.3.4 Configure (U) 11

Absorbs three data words from PREV and when all have been received, checks address; if does not match ours, acknowledges with own address, passes 4 words along to NEXT and performs **PUMP**.

Otherwise, sends acknowledgment (0000, xFFFF) and starts an update of the SPI flash variables. See above for timing.

3.2 VMOD Software

The VMOD has stringent timing requirements. The most demanding is propagation of the video pipeline down each module column with reliable signal integrity, no congestion, minimal delay through each VMOD, and minimal jitter in timing.

Each VMOD identifies the triads that belong to its module, tapping them off the pipeline in such a manner that there is no effect on pipeline timing or integrity. Each group of eight triads is assembled into a 24-bit value having eight bits each of Red, Green and Blue such that the first triad is the low three bits of Red and the last is the high three bits of Blue. These three octets are buffered in the GA144 and transmitted to the ARM at 4.5 Mbit/s according to the protocol outlined above.

These are the default continuous tasks of the VMOD board that operate as an autonomous, closed system. Additional functions such as Vertical Sync detection, refresh synchronization across modules, closure of prematurely terminated frames, and discarding of superfluous video are part of this process. Administrative functions such as watchdog timer maintenance, control and status communications, and auto-configuration take place in parallel with these tasks.

The following sections detail the functional elements of this software and show how they are organized physically and logically to fit within a GA144.

3.2.1 Chip Floorplan Diagram

The GA144 is deployed as depicted here:

GA144 Floorplan for VMOD

3.2.2 Parameter Storage and Retrieval

In the above floorplan there are several nodes requiring parameters that are only learned during the discovery and configuration process. Examples are module sequence number in the serpentine, used for addressing of commands, and pixel windowing parameters. For this application we have created a new facility in the boot stream, namely the ability to interpret a small table from flash at boot time and use it to scatter values to specific places in nodes' RAM before loading the rest of RAM content. The application may change values in this table and then make them effective by resetting the chip.

Space for two copies of this table resides in the last 8k bytes of the 128k boot area in the flash; origin is absolute byte address x1E000 where the normal table lies. A second copy of the table may exist at x1F000. The table consists of an 18-bit validity flag followed by a sequence of entries, each having three 18-bit words. The first word of an entry is either -1 denoting end of table, or is a node number in **nn** format (i.e. 717 is the upper right node of the chip.) The second word is a RAM address within that node, and the third word is an 18-bit value that will be stored at that address in that node. The validity flag is positive if the following table is valid, or negative if it is not. *In this structure a valid table may have zero entries.*

When burning the program into flash, the default table is written at x1E000.

The first part of the boot stream inspects the word at x1E000; if the value is positive the table starting 18 bits later is valid and is the one processed. If this value is instead negative the normal table is invalid and so the second copy at x1F000 (also flag followed by entries) is processed. This part of the boot stream then scatters the values in this table to the places in node memories specified. The second part of the boot stream loads node memory, stacks and registers as is normally done. If a node uses `/part` to skip over the place(s) in memory where value(s) were preloaded from the table, then the data from the table will have effectively been merged with the normal memory loads.

When a VMOD needs to alter a parameter due to discovery or to a command, it logically reads the table from flash into a buffer, modifies the data word(s) desired, and then erases and rewrites the table. To accomplish this robustly the following steps are taken:

1. Read flag at x1E000. If positive, skip to step 2.
    a. Read table from x1F000 into the buffer.
    b. Perform steps 4 and 5, then continue with step 2.
2. Read table from x1E000 into the buffer and modify the desired value(s).
3. Erase x1F000 and write the buffer (updated table) there.
4. Erase x1E000 and write all but the first word of the buffer into this area.
5. Write a positive value into location x1E000.

Because the condition detected in step 1 can only exist right after the chip is booted, that case is handled during initialization and the updated table is kept in RAM so that step 2 does not need to read it either.

3.2.3 Crystal Oscillator

This is a standard GreenArrays Mark 2 (high frequency) one-pin crystal oscillator; in this high frequency variant, four nodes are involved in initially exciting the crystal but only one of them (node 715) is necessary after excitation is successful. Node 715 subsequently "dribbles" the crystal by synchronously tracking its signal edges and applying as short a "nudge" as practical in the appropriate direction, as soon as practical after each edge.

3.2.4 Clock Distribution

Node 715's pin 17 is distributed in silicon as phantom wake-ups for nodes 709, 713 and 717.

In this application, node 717 tracks the oscillator and passes its signal at a full 18 Mega-edges per second down through wire to node 517, where it is made available through a port for the ARM Receive node 417, and drives a signal through pin 517.17 to 217.17 where node 217 makes it available through a port for the ARM Transmit node 317. The external signal continues to pin 317.17.

Node 300 uses this, and other information, to determine when the watchdog timer will be "tickled". It also prevents full operation until the 9 MHz crystal has been started successfully and is running stably.

On reset, 300 counts edges coming in on 317.17, toggling the watchdog every 10,000 edges. We count 900 such sets of 10,000. The first 800 are enough for all of the edges produced by an unsuccessful crystal start, and the next 100 are to ensure the crystal is running well by insisting on half a million full cycles. If the start-up fails, we stop toggling the watchdog and will soon reset. If we see all nine million edges then we believe that we have a good 9 MHz clock for transmission to the ARM and for receiving data on the control/status daisy chain. Accordingly, messages are sent to node 203 enabling the pixel pipeline, and to the control/status processing nodes enabling receipt of data from the preceding module.

Thereafter, node 300 divides the clock down by a factor of 30 (300 kHz, 600K 1.67µs edges/sec). The watchdog is toggled every 10000 of these edges (16.67 ms) and a data clock edge is sent to the PREV and NEXT daisy chain transceivers every 1.67µs for bit synchronization at 200kb/s. Once this is happening the VMOD will only reset if power is lost, if the 9MHz oscillator fails, or if the previous VMOD in the daisy chain sends a break signal as commanded by the VCTL.

3.2.5 Video Pipeline

Video comes down the column on four differential lines: D2, D1, D0 and half frequency clock. All four of these lines are driven simultaneously with no intentional skew. Clock skew to achieve safe set-up time is accomplished internally on the receiver, by delaying the sampling of the data lines after seeing the clock edge. The mapping of 24-bit pixels to the sequence of triads is as follows, top row being first transmitted:

Seq	D2	D1	D0
+0	R2	R1	R0
+1	R5	R4	R3
+2	G0	R7	R6
+3	G3	G2	G1
+4	G6	G5	G4
+5	B1	B0	G7
+6	B4	B3	B2
+7	B7	B6	B5

The critical path for column video passes through six nodes; these have no decision making to do. Timing is critical as is noise immunity. These nodes are as follow:

008: Detects the next clock edge on 008.17 and sends a phantom wake-up to 007 that triggers its reading the parallel **data** port. 008 intentionally delays about 25 ns before triggering the read to give set-up time for the incoming four data lines (clock and data). It also delays for a cycle time of 30-31ns so that it cannot give stimuli to node 007 faster than the pipeline can keep up with it; *this must always be the longest cycle time in the pipeline.* ***As it turned out 3/12/16, the higher temp in the room has apparently warmed the chip enough that the initial setting of this delay was causing us to miss triad clocks, so this is going to have to be adjusted and set further.*** On reset, this node delays for approximately 100 ms before looking at the incoming clock. The purpose of this delay is to prevent our ramming triads into the high speed part of the pipeline until the entire chip has been booted; allowing this pipleline to become congested destabilizes the application. After the time delay, normal operation begins as above.

007: This is essentially a wire node transferring from **data** to **rd-u** (nodes 006 and 107) which must be punctual or they may lose program control. The multiport writes by 007 and 107 are protected by the long cycle time of node 008. This node cycles in 16.75ns. On reset, we passively synchronize with the first several signals from node 008 before commencing normal operation as above; the purpose of this is to make certain we aren't going to begin operation by short-cycling the taps.
107: This is a wire with a tap to 106, transferring from **down** to **r-l** . Cycle 13.0ns.
108: Standard wire. Cycle 13.0ns.
109: Standard wire. Cycle 13.0ns.
009: Uses a lookup table to translate the received 4-bit value to its 8-bit equivalent as four differential signals. Cycle is 28ns.

Transit time for a triad down a module column is on the order of 61ns per VMOD with 3' cable.

3.2.5.1 Vertical Sync Detector Tap

Node 006 is a wiretap of node 007 that serves as a punctuality buffer. It feeds node 005, a state machine that monitors for the start and end of image frames. The end of a frame is detected by a long time with no triads. The start is indicated by the first triad after an end. Detection is done by polling **io** in a loop whose cycle time is typically 15.3ns. When the loop sees node 006 writing to us, we read the word and reset our counter. When the count value expires we write a negative number to the video switch, with critical timing requirements. The count value does not require great precision:

For 1080p video, a pixel time is 6.734ns, a line is 2200 pixel times, and a frame is 1125 line times, of which 45 are blanked around vertical sync leaving 1080 for video. Thus a line time is 14.81µs and 45 of those are 666µs. A good setting for the delay loop to detect end frame would be 200µs; that gives plenty of dead-band for variations in loop timing and also gives us plenty of time, roughly 400µs, to top off a delay line (worst case if it's empty is 1920 words at 50ns each, or 96µs).

The VS node runs at a high duty cycle because it must loop and poll for all but 400µs of each frame.

3.2.5.2 Triad Data Tap

Nodes 106 and 105 are slow wires tapping node 107, each with measured delay of >19ns above that of a perfect wire, and cycles on the order of 28.8ns. They feed node 104 which uses a lookup table to convert the scrambled 4-bit values from node 007 to binary values 0..7 by reversing the low order three bits and ignoring the clock, and these are fed directly into a port on the video switch node. Time delays are designed so that signals can propagate from node 007 through wire 006 and the VS node to the switch faster than they can make it from 007 through 107, 106, 105 and 104.

3.2.5.3 Video Switch Node

This node normally receives triads through the data tap, but it does so with a multiport read so that it may also learn about the end of a frame from the vertical sync detector by reading a negative value. The timing of the two taps, data and sync, ensures that the switch will never receive data from more than one of these at a time.

In general its task is to insert a start frame marker when the first triad of a frame is received, and then to pass video from the first triad in a frame until end of frame (VS) is detected, at which time it pads out the assembly pipeline to an 8-triad (pixel) boundary, inserts an end frame marker, and waits for the start of the next frame.

The switch is a state machine. Initially it is waiting for a SOF (start of frame) marker, x30000, from the VS node. This marker is sent to us by VS before the first triad has had time to make it through all the delays, and it is passed through the pixel assembly pipeline to trigger transmission of the previous frame into the module; this ensures that all modules will start receiving that frame within a line time plus propagation of the first triad down a full column. Thereafter the switch passes triads through, counting them modulo 16 (two pixels) until it sees an EOF (end of frame) marker, x3FFFF, from the VS node. At that point the switch transmits [1..16] zeroes onward, to flush the pixel assembly pipeline to the next word boundary, before sending the EOF onward to close out the current frame, switch delay lines and prepare to receive the next frame. It also resets all the counters and states along the way. After the EOF marker is passed, we repeat this paragraph, waiting for the next SOF marker.

We may want to reconsider the above behavior in concert with the VS node to make the system more robust at ignoring noise during vertical blanking, for example by looping for 100µs, during which time any incoming noise data are received and discarded, but this complication is omitted intially. In this scenario we would not get an SOF from VS but simply pass along the first triad received after the time delay contemplated here. However we'd still have to push the transit time very near the 30ns limit in order to send the SOF marker ahead of the first pixel.

The delay between the vertical sync detector's reading **io** that shows no write from 007 the last time, until we have written the value into the switch is 34.3ns. This is also as good as seeing that node 107 was not writing at that time; there is one wire delay between 007 and the port into 005 just as there is such a delay between 007 and the port into 106. Because the switch is doing a multiport read it is imperative that a triad that was coming in just as we were reading io cannot possibly make it to the switch before we do. The time delays in the triad tap path above guarantee this conservatively.

Each slow wire is a measured delay of 19ns over a perfect wire, with a safe cycle of 28.8ns. The look-up table node delays about 12ns over a perfect wire with a cycle of 27.6ns. Thus we have a total delay of more than 51ns (plus 3 wire delays) from the wall port of 107 to the switch, while the time from last **io** read by VS to its switch port is on the order of 35ns. This should be more than safe.

3.2.6 Pixel Counting and Assembly

Once we have vertical synchronization, we can confidently isolate the pixels relevant to our module and subject them to further processing. We want to escape the fatiguing <30ns cycle time constraint, with its associated consumption of nodes and energy, as early as feasible.

3.2.6.1 Pixel Windowing

The best place to filter only those pixels for our module is immediately after the switch, when we are confident of frame boundaries. Node 003 serves this purpose, counting down **pxofs** pixel pairs from top of frame then processing **pxmo** + 1 pixel pairs before discarding any that follow. In addition it recognizes and passes along frame markers when they occur as expected on mod 16 triad boundaries. These two parameters are provided by the flash variable table during booting.

3.2.6.2 Byte Assembly

Node 103 assembles sequences of 16 triads into sequences of 6 bytes (2 pixels) and passes these along, recognizing and passing frame markers on mod 16 triad boundaries. Cycle timing varies due to the coprime data sizes, but it proved feasible to balance the cycle time to a (calculated) range of 13.2 to 22ns. As it turned out there was no timing benefit to checking for markers every two rather than one pixel, but the extra code does save energy so it was retained. This is the last node whose cycle time is limited to <30ns.

3.2.6.3 Word Packing

Node 203 receives start and end frame markers surrounding even sequences of bytes. The former are passed through without change while the latter are packed into 16-bit positive words. The first byte of each pair goes into the low order half of the word because our transmission order is LSB first.

On reset, this node receives and discards all incoming markers and bytes until it sees the message from node 300 indicating stable clock for ARM transmission. It then continues discarding input until it receives, and passes, the first SOF marker; thereafter normal operation proceeds as above. This is necessary because the pixel pipeline can be congested, carrying clear back through the triad taps, if we attempt ARM transmission during the start-up sweeps because it includes bogus frequencies as well as long delays waiting for a reaction from the crystal.

3.2.7 Frame Buffers

The largest module frame of interest is 1280 24-bit pixels and therefore 3840 octets of data. Two buffers are implemented as separate 30-node delay lines. Each delay line node contributes 128 bytes of storage and there is no code required in either RAM or ROM.

3.2.7.1 3840-byte Delay Line

These delay lines are synchronous FIFOs, which is to say that they are always perfectly full and that a word becomes available at the output only after a word is inserted at the input. The two-pointer asynchronous FIFO model does not apply in this design. This delay line mechanism words at a sustained rate of approximately 52ns per word; this is sufficiently fast given that our incoming pixel data arrive at 213ns per word (assuming that **PX** is 40 and **MX** is 48). Special management code at the entry (Tx Helper) and exit (Rx Helper) of each delay line arranges that it may be filled and later emptied without attention from outside the 32-node assembly except for the actual insertion of data at the Tx end and its removal at the Rx end.

3.2.7.2 Pipe Selector

To support near simultaneous initiation of frame changes in the modules (Less than 20µs; on the order of a line time, 14.81µs, plus 45 61ns VMOD propagation delays (61ns) for another 2.8 µs), we double-buffer the frame images. At the source end, node 205 serves as a pipe selector. On boot, delay line A is selected to hold the next incoming frame. The following frame goes into delay line B, and the two alternate forever.

When a start of frame marker is received, we pass it to both the old and the new delay line Tx nodes. On receiving it, the old delay line begins pushing the previous frame out to the ARM (except for the first marker given to delay line which is at this time empty), and the new delay line begins absorbing the next frame. These activities are synchronized by complementary actions at the delay line Rx nodes. When an end frame marker is received, that is passed to the delay line currently being filled and we switch the roles of A and B.

3.2.7.3 Delay Line Tx Helper

A delay line Tx node waits for a marker (SOF), then copies in all words received until it reaches 1920 or sees a marker (EOF). If less than 1920 words were received, the buffer is padded with trailing zeroes. The Tx node then waits for another marker (SOF) that signals it to begin pushing out exactly 1920 words by inserting that many zeroes, creating "pressure". No markers are included in the delay line because this is not necessary; the counts are always exact.

3.2.7.4 Delay Line Rx Helper

The delay line Rx node begins by sucking and discarding 1920 words, creating a "vacuum" so that the delay line can be filled to capacity. It then waits for the first word to emerge from the delay line, precedes it with a command marker, and transmits all 1920 words to the ARM regardless of module configuration.

3.2.7.5 Pipe Demultiplexor

This node simply transfers 1921 words from delay line A Rx node, then the same number from B, and so on.

3.2.8 ARM Serial Nexus

Our main concern with the ARM is to give it streams of pixel data with reliable (and controllable) timing so that the ARM can apply strategies to avoid visible interference with its internal multiplexing cycles. This is handled by a multiplexor which manages delivery of the pixel streams but allows transmission of very short commands during the vertical blanking interval, and a serial transmit node that operates at a crystal synchronized rate of 4.5 Mbit/s.

3.2.8.1 ARM Transmit Multiplexor

Node 316 waits for data from the Pipe Multiplexor. When such is received, markers are translated into *Frame Commands* and subsequent data are transmitted as pixel data. The frame command used is supplied by the flash variable table during booting, and encodes the orientation of the module's LED array for mapping of video rows to the correct LEDs.

The pixel data are counted, and after sending 3840 bytes we are presumed to be in a vertical blanking interval. *Thus, if we ever need to implement additional commands, IO can polled and if there is a command waiting from node 416 we can pass it to the ARM, returning afterward to the wait for the next frame to arrive from the Pipe Multiplexor.*

3.2.8.2 ARM Transmit Node

Node 317 supports transmission of single and double bytes with data parity, and 3-byte sequences for commands as described in the protocol description above. To facilitate solid pixel timing, the COM port may be "captured" by any source that promises to deliver characters fast enough to avoid missing incoming clock edges.

3.2.8.3 ARM Receive Node

This node may be used for replies to future ARM commands if necessary.

3.2.9 Watchdog Timer

The watchdog timer is kept happy (or not) by node 300 as described above in the clock distribution section.

3.2.10 SPI Flash Management

Five nodes provide the ability to manage the variable table in flash. 705 handles basic SPI operations and spin for completion of writes; 704 supports flash operations like write enabling, table erase, table read, and bit stream write using byte write only for compatibility between old and new flashes. 703 maintains a local copy of the variable table in flash and provides port executable commands to read, write and erase either of the table areas on flash. This table always reflects the current content of flash, not the running system.

Node 702 provides high level operations to maintain the values and integrity of the variable table on flash. On reset, node 702 reads the table from flash and if an incomplete update is found the table is fixed. This is done by inspecting the flag at x1E000. If it's negative then the updated table must exist at x1F000 and is copied down safely. Thereafter the table in node 703 RAM is maintained as equivalent to the table at x1E000 on flash. Port commands are provided to read and write the four variable values, updating flash as appropriate.

Node 701 is the final interface to command processing. It maintains two local copies of variable values, those in flash (set by commands from VCTL) and those currently running (reported in muster.) Only the first, VMOD address in the serpentine, may be changed without rebooting the GA144. This node supports three functions via port commands from node 601:

Store address into flash image and burn.

Store video parameters into flash image and burn, orientation received first.

Read video parameters currently running, orientation returned last.

3.2.11 Control/Status Comunications

The PREV and NEXT RS485 lines each use two nodes, one for the transmit/receive pin and basic serdes functions, the other to manage the direction line and handle higher level message oriented operations. For PREV, 500 is T/R and 600 is control; for NEXT, 200 is T/R and 100 is control. Node 400 functions as wire, carrying clock edges from node 300 to 500.

As indicated earlier, data are framed as 16-bit characters, MSB first, no parity with two stop bits. Each bit time takes three of the incoming clock edges so that our data rate is 200 kbaud.

On the PREV channel, node 600 resets with the line in receive mode but waits for a signal from 601 indicating that the 9MHz time base is stable. It then handles outbound messages far enough that only legitimate ones make it to node 601 for processing. All data for a given message are received before anything is done with them, so that timeouts are silently ignored. When a good command has been received, **func** in 601 is called and three words (command, address, command shifted left 2 bits) are sent through the port following this call. If the command was configure, the three parameters are first pushed onto node 601 stack with the first parameter received (orientation command) on top of stack. Thereafter node 600 expects to pass data inbound on PREV until a word of xFFFF has been sent. In 600 !w3 is actually a w2 and return; this makes the function take three clocks but each leaves the timing from 300 made happy, necessary so that we can correctly transmit the incoming words whenever they arrive.

3.2.12 Command and Control Processing

func in 601 handles the six cases: Muster, Enumerate, and unicast messages (Reset for us, Reset for downstream, Configure for us, Configure for downstream). In the cases of Reset for us and Configure for us, node 601 returns a two-word acknowledgment (our address followed by xFFFF) and then does that is requested (resets the NEXT VMOD or stores video parameters in flash.) Otherwise unicast messages are simply copied to NEXT and we **pump** replies back from NEXT to PREV until we see a timeout (passed back as xFFFF) or an xFFFF from downstream (meaning no more data to come.) If the unicast message Muster, Enumerate, node 601 generates the appropriate acknowledgment or feedback from our module. In the cases of broadcasts (enumerate or muster) we reply to NEXT with our address or our three video parameters followed by address as requested, update our address for enumerate, and pass the command on to NEXT after which we **pump** as above.

Node 701 handles the flash operations for 601 as noted above.

Specialized wire carries the traffic to and from NEXT between 601 and 100. Node 100 manages the exchanges. The first word sent down the wire is **n** followed by **n+1** words to transmit on NEXT. The line then turns around and moves data until xFFFF is passed after which a new n is expected. If **n** is negative then the message consists only of **n** with no replies and it means to reset the NEXT module. There are three versions of the wire code because node 300 must signal the PREV receiver when it's determined that the 9MHz oscillator is stable and that PREV has a stable time base available.

3.2.13 Communications to/from NEXT

Node 100 implements the same logic as do the specialized wires. The difference is that the break operation, or the exchange, take place on the NEXT serdes rather than being passed to the next node through a port. As noted above, each input word from NEXT times out in 600µs.

3.2.14 Loose Ends

Items to revisit sooner or later:

- Node 600 is now safe, leaves timing tended while waiting for incoming words.
- If room still exists to do so, have **ent** in node 600 wait till it can get a timed out receive to avoid starting to listen while a character is incoming.
- Optimize ARM interrupt code for non-interference with multiplexor timing (done by Dean)
- Close loop around crystal start-up tighter than just using the watchdog
-

3.3 VCTL Software

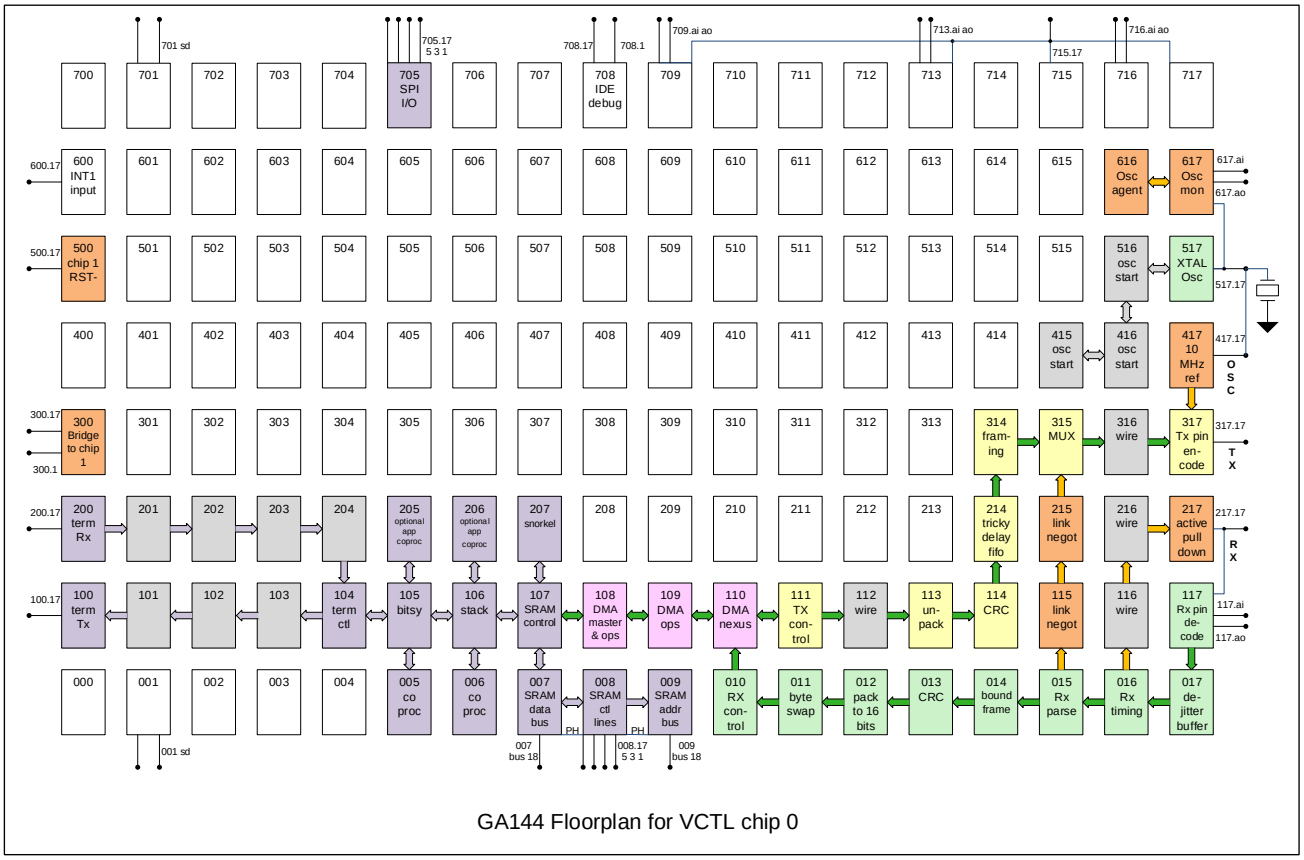
The VCTL is principally concerned with high level supervisory tasks, such as initializing the FPGAs, initializing and managing the ADV7611 video chip, controlling the configuration of the overall display and monitoring the status of each VMOD in the display.

The polyFORTH environment is ideal for this sort of work. Two GA144s are included; chip 0 runs polyFORTH with ethernet capability and handles some of the I/O interfaces, with chip 1 acting as a slave to provide additional I/O.

The following sections detail the functional elements of this software and show how they are organized physically and logically to fit within the GA144s.

3.3.1 Chip 0 Floorplan Diagram

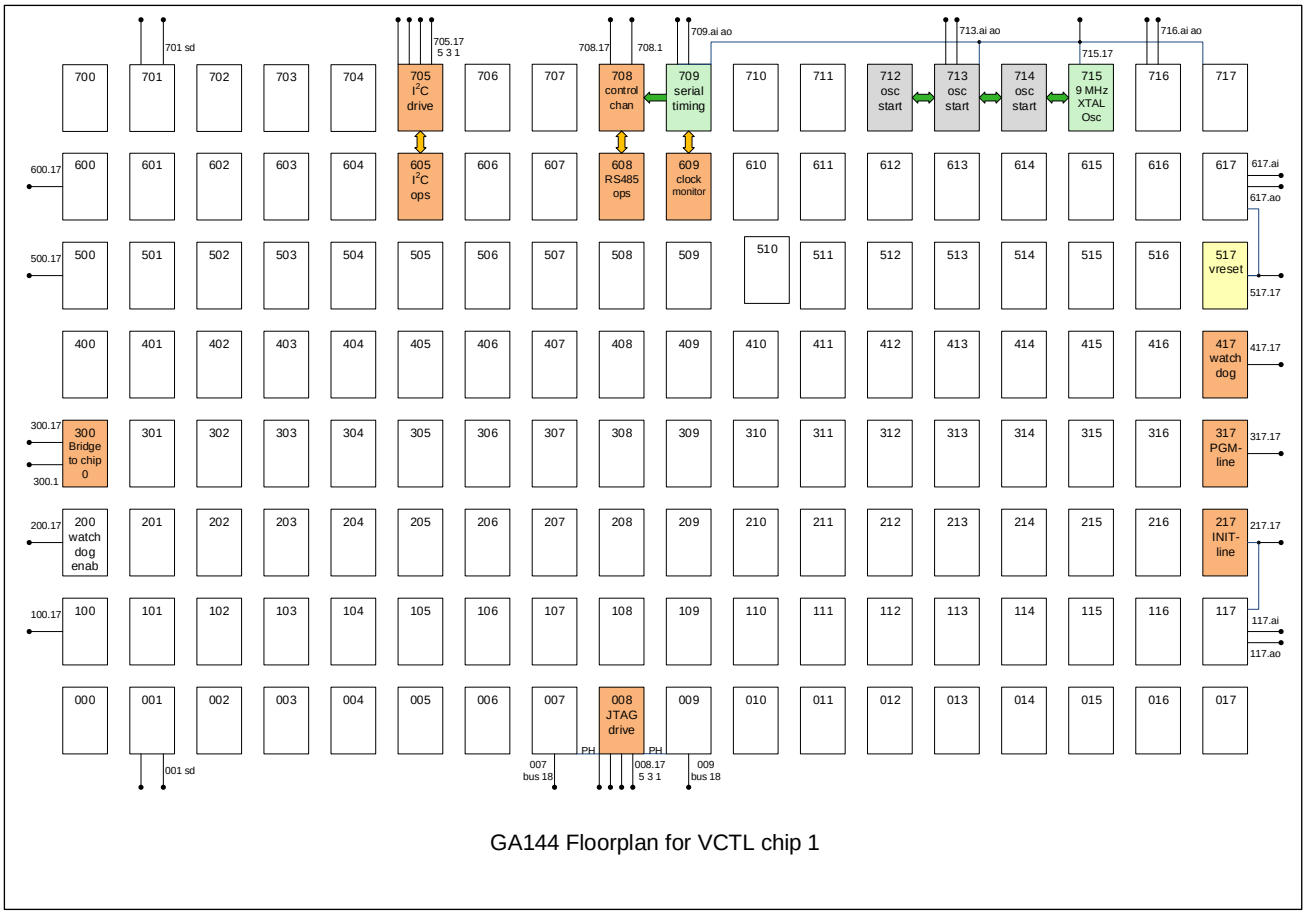
The main GA144, chip 0, is deployed as depicted here:



GA144 Floorplan for VCTL chip 0

3.3.2 Chip 1 Floorplan Diagram

The expansion GA144, chip 1, is deployed as depicted here:



GA144 Floorplan for VCTL chip 1

3.3.3 ADV7611 Control (I²C)

This code, all written in polyFORTH, initializes and activates the ADV7611 for the 1080p operation we desire. Much of this procedure is documented separately in AN018.

3.3.4 FPGA Configuration (JTAG)

polyFORTH is responsible for initializing and configuring the FPGAs using JTAG. We captured the entirety of JTAG traffic generated by the Xilinx iMPACT program, to prove that we knew how to make a configuration .bit file that iMPACT did not modify in any way during its transmission. It turned out that this required having bitgen set the stream up to use TCK as startup clock; the actual statement used is as follows:

```
bitgen $(intstyle) -g UserID:0x09470947 -g DriveDone:yes
      -g StartupClk:JtagClk -w $(project)_par.ncd $(project).bit
```

3.3.4.1 Bitstream by Xilinx iMPACT

As it turns out, iMPACT is very sloppily written and does a lot of unnecessary things during its JTAG sequence. These are illustrated below.

```
JSIZE RUN

      0 RESET
      8 RESET  Redundant.

     41 Instr(24) 11 IDCODE 111111 111111 111111 100100
     81 Data(35) Bit stream:
                    Out: 11100000000000000000000000000000
                         000
                     In: 00011001001000010000000000000100
                         100  24001093 as documented.  Three bits in front
                              are to push three bits out of chips 1, 2, 3.

     84 RESET  All of this is redundant:
    141 Instr(48) 77 BYPASS 010101 010101 010101 010101
                            111111 111111 111111 111111
    144 RESET
    197 Instr(24) 11 IDCODE 111111 111111 111111 100100
    237 Data(35) Bit stream:
                    Out: 11100000000000000000000000000000
                         000
                     In: 00011001001000010000000000000100
                         100

    267 Instr(24) 77 BYPASS 111111 111111 111111 111111
    297 Instr(24) 13 JPROGM 111111 111111 111111 110100
                                Effectively equivalent to asserting PGM-
                                INIT- asserts about 100ns after entering
                                Update-IR on this command, stays low about
                                287.5µs; presumably we must spin on it.

    327 Instr(24) 05 CFG_IN 111111 111111 111111 101000
    330 RESET                   What is the point of doing this?

    363 Instr(24) 05 CFG_IN 111111 111111 111111 101000
    416 Data(48) Bit stream:
                    Out: 11100000 00000000 00000000 00000000
                         00000000 00000000
                              Although this is 3 16-bit words it is not
                              legitimate frame format per the manual.  ???
                     In: 00000000000000000001110000000000
                         0000000000000000

    446 Instr(24) 05 CFG_IN 111111 111111 111111 101000
2725363 Data(2724912) Octets, front only:

000   E0 00 30 A1 00 08 00 00 00 00 FF FF FF FF FF FF   |`_0!____________|
            <-High Z ->  but delivered without synchronization!
010   FF FF FF FF FF FF FF FF FF FF AA 99 55 66 30 A1   |__________*_Uf0!|
      <---------- dummy ----------> <-- sync -> <reset
```

```
020   00 07 20 00 31 A1 03 80 31 41 3D 06 31 61 09 EE   |__ _1!__1A=_1a_n|
       CRC> <NOP> <frameleng> <set COR1 > <set COR2 >
030   31 C2 04 00 10 93 30 E1 00 CF 30 C1 00 81 20 00   |1B____0a_O0A__ _|
      <-wrt/chk devID-> <Mask CTL-> <-wrt CTL->  nop
040   20 00 20 00 20 00 20 00 20 00 20 00 20 00 20 00   | _ _ _ _ _ _ _ _|
       nop   nop   nop   nop   nop   nop   nop   nop
050   20 00 20 00 20 00 20 00 20 00 20 00 20 00 20 00   | _ _ _ _ _ _ _ _|
       nop   nop   nop   nop   nop   nop   nop   nop
060   33 81 3C C8 31 81 08 81 34 21 00 00 32 01 00 1F   |3_<H1___4!__2___|
070   31 E1 FF FF 33 21 00 05 33 41 00 04 33 01 01 00   |1a__3!__3A__3___|
080   32 61 00 00 32 81 00 00 32 A1 00 00 32 C1 00 00   |2a__2___2!__2A__|
090   32 E1 00 00 33 A1 1B E2 33 C2 00 00 00 00 20 00   |2a__3!_b3B____ _|
0A0   20 00 30 22 00 00 00 00 30 A1 00 01 50 60 00 02   | _0"____0!__P`__|
                                        <Wrt to frm
0B0   98 AD 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |_-______________|
      data>
0C0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
0D0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
0E0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
0F0   00 00 00 00 10 00 00 00 10 00 00 00 00 00 00 00   |________________|

2725393 Instr(24) 14 JSTART 111111 111111 111111 001100
2725447 Instr(24) 05 CFG_IN 111111 111111 111111 101000
2725596 Data(144) Bit stream:
                      Out: 11111111 11111111 11100000 00000000   FFFF E000
                           00000000 00000000 10101010 10011001   0000 AA99
                                                sync     sync
                           01010101 01100110 00101001 00000001   5566 2901
                             sync     sync   <read stat 1 wd->
                           00100000 00000000 00100000 00000000   2000 2000
                                                                  nop  nop
                           00100000 00000000                     2000
                                                                  nop
                            In: 000000000000000000011111111111111
                                111111000000000000000000000000000
                                000101010101001100101010101100
                                110001010010000000100100000000000
                                0000010000000000
2725626 Instr(24) 04 CFG_OU 111111 111111 111111 001000
2725650 Data(19) Bit stream:
                      Out: 1110000000000000000
                       In: 000 0011 0111 0011 1000
                           15                   0   Valid STAT register!!!  x3738
2725653 RESET
2725686 Instr(24) 05 CFG_IN 111111 111111 111111 101000
2725819 Data(128) Bit stream:
                      Out: 11100000 00000000 11111111 11111111   E000 FFFF
                           10101010 10011001 01010101 01100110   AA99 5566
                           00110000 10100001 00000000 00001101   30A1 000D
                           00000000 00000000 00000000 00000000   0000 0000
                                Note that main bitstream seems to end with
                                the 30A1 000D pattern too, which means DESYNC.
                            In: 000000000000000000011100000000000
                                000111111111111111111010101010011
                                001010101010110011000110000010100
```

```
                        00100000000000000000000000000000

2725849 Instr(24) 77 BYPASS 111111 111111 111111 111111   Why do we do this again???
2725879 Instr(24) 77 BYPASS 111111 111111 111111 111111
2725882 RESET
2725920 Instr(24) 14 JSTART 111111 111111 111111 001100
2725974 Instr(24) 77 BYPASS 111111 111111 111111 111111
2726004 Instr(24) 77 BYPASS 111111 111111 111111 111111
2726013 Data(4) Bit stream:
                      Out: 0000
                       In: 0000 ok
```

3.3.4.2 Our Bitstream

We eliminated the chaff from the above procedure and were finally able to integrate the configuration files James Bowman produced for us into a working JTAG bitstream for initializing the FPGAs. As it turned out we were able to configure all four chips, whereas the Xilinx tool never did successfully configure all of them. The limitation was number of chips configured, not position on the daisy chain. We saw no reason to analyze that problem further because we had no further need of iMPACT anyway.

3.3.5 FPGA Parameters (I^2C)

After the FPGAs have been programmed using JTAG, they present a register interface on the I2C bus for configuration of the overall display. For each column, its starting position in the X axis must be set in pixels relative to zero at the left side of each line; and for each column the width of that column in pixels must be set. For the FPGA as a whole, the Y axis position of its top row must be set in rows relative to the top row of the source image.

3.3.6 Command Channel (RS485)

A 9 MHz crystal oscillator is maintained by node 10715 after starting via 10712, 13 and 14. The signal from this crystal is monitored via a shared pin by node 709 which divides it down by 180 to 50 KHz, yielding one edge every 10 µs. This sequence of 10 µs stimuli is presented to nodes 708 and 609.

Node 609 acts as a monitor for crystal operation and may be interrogated by polyFORTH using minimum Ganglion transactions as a watchdog criterion. The expected transaction consists of two outbound words (focusing call and one data), and one inbound word (data).

Node 708 serves as a transceiver for the RS485 command channel, performing functions commanded by polyFORTH and implemented by node 608 using Ganglion transactions.

3.3.6.1 Node 708 RS485 Transceiver

This node transmits 8 or 16 bit values and receives 8 bit values under crystal control at 33.333 kb/s.

Transmit bit times are three 10µs clock events long. Framing is 8 data bits, no parity, one stop bit.

The receiver samples for start bit on clock events so we start]0..10] µs into the start bit. We wait one clock after that so we are now at]10..20] into the start bit, then sample eight data bits at 3-clock intervals from there. After sampling last data bit we wait two clock events which places us]0..10] µs into the stop bit. Consecutive characters may be received if we start looking for the next well within the 20µs of remaining stop bit. Each receive operation is given a timeout value in 10µs units, limiting the amount of time we will wait for a start bit. The character value returned in the case of a timeout identifies it as such.

3.3.6.2 Node 608

Under ganglion control, this node performs high order functions on the RS485 line. polyFORTH functions are:

xchg performs an RS485 exchange operation. It's followed by arguments **tn t0 ni no** after which it expects **no+1** words to send, then returns **ni+1** received words. The timeout on the first receive word is **(t0+1)*100**µs; on the remaining words, **(tn+1)*100**µs. After a timeout, or a received xFFFF value, any remaining receive words are synthesized as xFFFF. For diagnostic purposes, an actual timeout detected by the VCTL itself is returned as xFFFE.

break transmits a break signal for 10ms on the line. Expects five words (ignored) and returns one.

3.3.7 System Booting

We will be using polyFORTH in auto-boot configuration. In this configuration, auto-baud is started but if it is not completed in a short while we proceed to load block 9 without waiting for it. If the auto-baud does complete in a reasonable period then we do not load block 9 but instead wait for human input.

3.3.8 Watchdog Strategy

The main concerns after reset are successful start-up of the 9 and 10 MHz crystals. polyFORTH functions are:

?9MHZ returns normally if interrogation of node 10609 indicates that the 9 MHz oscillator is running. Hangs if not, resulting in watchdog timeout.

3.3.9 Sign Configuration Specification

The FPGAs and the VMODs must be configured properly to distribute video through the display geometry of this particular sign. The configuration is done dynamically, and its correctness is enforced periodically, in a fully automated manner. This process begins with a sign configuration block written in the sign configuration language. Syntax and semantic rules must be respected in the configuration block. Here are three examples of such blocks:

For a plain vanilla 1080P sign built from 24x40 modules we would have this:

```
318
 0 ( CONFIG of this sign)
 1 ( Module geometry)  45 48 GRID  !INDICES
 2 ( Column widths)  40 WIDE  !PIXWID
 3 ( Modules)  24 40 0 MODULES  !VPOS
 4 ( Sign window)  0 0 WINDOW
```

This illustrates all of the standard vocabulary needed to define homogenous signs made of a single module in a single orientation (Required words are shown in red):

GRID (y x) sets the maximum dimensions of the sign in modules. Some columns may have fewer modules than the value of **y** but none may have more. The values shown are appropriate for a 1080x1920 display built of 24x40 modules.

SHORT (n x) indicates that column **x** is an exception with fewer modules, **n** , than the **GRID** phrase declares.

!INDICES is used after the number of columns and number of modules in each column have been declared using the words above, to define the transformation between physical positions on the sign and addresses on the RS485 control and status daisy chain whereby the VCTL controls the VMODs. The following may be used after **!INDICES** ...

WIDE (n) sets default column width to n pixels. *Note that the FPGA design, and for that matter system timing design, requires that column widths must be less than or equal to 40.*

DIFFERS (n x) indicates that column **x** is an exception with a different pixel width, **n** , than the **WIDE** phrase declares.

!PIXWID is used after the pixel widths of all columns have been declared using the words above. The following may be used after **!PIXWID** ...

MODULES (y x rot) sets the default size and orientation of all sign modules. rot is an angle (0, 90, 180 or 270) indicating the angle it has been rotated counterclockwise, as seen from the front of the module, from the standard orientation (RJ45 connector openings point downward). y and x are the dimensions of the module seen with rotation zero; standard Sunrise modules, when not square, have x as the larger dimension.

MCOL (y x rot n) overrides the current settings for column **n** to be populated with a different type..

MODULE (y x rot i) overrides the current settings for *module index* **i** to be populated with a different type.

!VPOS is used after the sizes and orientations of all modules have been declared using the words above, to define the vertical starting positions and sizes of each module in its column. The following may be used after **!VPOS** ...

SKIP (x) indicates an FPGA column (0..59) that has been skipped over in wiring the sign. No video will be generated for that column; the video that would have been displayed in it is shifted right to the next column.

WINDOW (y x) defines the upper left pixel origin of the displayed window relative to the incoming video. This affects only the FPGAs, based on all the above information, so it is implemented very quickly and may be changed dynamically without restating any of the above information.

For the sign used on the bench at GreenArrays we would have this:

```
318
 0 ( CONFIG of this sign)
 1 ( Module geometry)  3 4 GRID  1 3 SHORT  !INDICES
 2 ( Column widths)  40 WIDE  ( 24 0 DIFFERS)  !PIXWID
 3 ( Modules)  24 40 0 MODULES  ( 24 40 90 0 MCOL)  !VPOS
 4 ( Unused columns)  3 SKIP  4 SKIP  5 SKIP
 5 ( Sign window)  7 3 WINDOW
```

The display is only 3x4 modules of 24x40 pixels in normal horizontal orientation (72x160 pixels total), however because column 3 doesn't work on this prototype VCTL we skip over three columns so that column 3 of the display is plugged into column 6 connector on the VCTL. Additionally, only one module (the top one) is populated in column 3.

The commented phrases on lines 2 and 3 would specify that column zero consisted of three modules rotated 90 degrees so that the column will only be 24 pixels wide (and 120 tall). The column could be shortened by one module using the phrase **2 0 SHORT** however that would also require removing the bottom module of column zero from the daisy chain so that the addressing would work correctly.

3.3.10 Configuration Management

While the FPGAs are maintained dynamically as the sign configuration is being processed, VCTL configuration may take far longer in a new sign or in one that's been reconfigured. Accordingly this process occurs in the background and is started after the configuration has been processed.

Periodically, they system performs a function named **CHECK** which takes a muster of all VMODs and their currently operating configurations (in terms of vertical position, size, rotation and address on the daisy chain.) Normally all VMODs will report, and their reports will reflect the correct settings of these parameters which are saved in flash at each VMOD so that the sign will power up as it was most recently configured.

The first thing **CHECK** does is to inspect the reply and determine whether any of the VMODS responding to the muster were improperly addressed. If any were, this must be corrected before we do anything else because we cannot successfully unicast a command until the addressing is correct. So in this case we enumerate the sign, causing all the VMODs to update their addresses in RAM and in flash.

If all the addressing was correct we then see whether any module did not reply to the muster. It is impossible to communicate with a non-replying module or with any that follows that module, so we need to pay attention to this. At present we reset the first non-responding module, assuming that it will have come fully up and running by the next time we **CHECK** . Because modules may be reset by their watchdogs, or may be in the process of restarting after a commanded reset to implement new parameters after changing flash, we may have to add a persistence check in this logic.

Assuming that all VMODs replied, and all are correctly addressed, **CHECK** next compares each VMOD's reply with the current sign configuration. Any differing VMODs are marked, and after checking all of them if any differ we update the flash in each module, in daisy chain order (nearest to VCTL first.) After starting all the updates, we wait long enough for the farthest VMOD to have finished updating its flash and then we reset all the updated modules in the reverse order (farthest from VCTL first). This minimizes the time required to update more than one module. The resets are needed to actually implement the configuration changes, which are stored only in flash when updated but which are not inserted into running code until the next flash boot.

Under ideal conditions with all VMODs working well, only two passes of **CHECK** should be required to fully configure a sign after it has been assembled from new or reclaimed VMODs.

4. Test Plan

When using a Windows PC as video source in critical testing, visual artifacts should be killed. Ctlpan/display/adjust clear type text: Turn it off. Ctlpan/perf options/adj visual effects: Turn off smooth edges of screen fonts. Display personalize: Select Windows classic style instead of Aero.

4.1 Control PCB

The testing addresses major sections in order, and will necessarily include some software development. Before testing a board assign it a serial number if populated. ***Maybe send a PCB to James before starting. He can power just the FPGAs and verify good communication with them over JTAG.***

4.1.1 Power Supply Section

- Power supply shorts: Pull all the 9xx jumpers and check for shorts to ground and between all supply buses. First on an unstuffed board and then on a stuffed one. Pins used are the center pins of each jumper. Yellow means tested OK, pink is shorted unstuffed; resistance values measured on stuffed board, orange is suspicious but not shorted in the unstuffed board.

Neg Pos->	GND	J902	J903	J904	J905	J907	J908	J909	J910	J911	J912	J913	J914
GND													
J902 VCD_3V3													
J903 VXIO_3V3													
J904 VXAUX_3V3													
J905 VXINT_1V8													
J907 VAC_1V8													
J908 VAP_1V8													
J909 VAD_1V8													
J910 VAT_3V3													
J911 VAOP_3V3													
J912 VGA_1V8													
J913 VGA_3V3													
J914 VGA_5V													

- Power each section up (external power off, add jumpers, then external power on; do NOT change jumpers while external power is energized) before testing it below. Check voltages and note any discrepancies.

4.1.2 Computer Section

- No Boot jumper in, disable both SPI flashes (J101 1-3, 2-4), Ser Boot jumper in, all power jumpers out. Insert J203 1-2 and verify low (if not ground before proceeding).

- Power up each of the three VGA_ sections and check voltages.
- Plug in USB ports A and B, configure each FTDI chip per EVB001 instructions but with different data (see the commissioning section below). Verify the 3.3V test points.
- Power all three VGA_ supplies and verify voltages. Verify reset button at J106 pin 1 and J203 pin 3.
- IDE into port A and verify basic communication and ability to control the reset line (Raw, J106 (J203 pin 3).
- Run **selftest** on chip 0.
- Run **autotest** without SERDES to test chip 1.
- Boot basic (non-Ether) polyFORTH (430 load) and verify it runs using FTDI port B.
- Run SRAM tests from polyFORTH.
- Enable U102 (J101 3-5) and verify ability to program and boot from flash.
- Devise a test procedure for module control daisy chain drivers.
- Verify the testpoints for nodes 715, 10100, 10500, 10600, 10709.

4.1.2.1 New hardware and code (NOT CRITICAL PATH)

- Reconfigure for Ethernet (with crystal excitation) and test circuitry and code.
- Reconfigure polyFORTH for W25Q128 SPI Flash and verify it works.
- Test Charley's new arrayForth code for writing boot to flash using "modern" 256-byte writes.
- Test Watchdog enable and circuit.
- Get 9 MHz crystal osc going for control channel.
- Define, develop and test control protocol.

4.1.3 ADV7611 Section

- Power all five VA_ supplies and check voltages.
- Verify I^2C operations as previously bread boarded (slight code change)
- Initialize the chip

  What I see for registers after +ADCHIP on first try:

```
0 io .DEV
    000   1E 05 F2 40 62 28 A6 40 14 00 90 44 42 1E 0F 1E   |__r@b(&@___DB___|
    010   00 00 81 0D 6E B0 43 5A 34 00 02 00 00 00 00 00   |____n0CZ4_______|
    020   F0 00 00 03 00 00 00 00 00 00 00 00 00 00 00 00   |p_______________|
    030   88 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
    040   E2 30 08 00 00 00 00 40 00 00 00 00 00 00 00 00   |b0_____@________|
    050   00 00 00 00 00 00 00 00 00 00 00 04 00 00 00 00   |________________|
    060   00 00 00 00 00 40 00 00 00 00 00 00 00 00 00 00   |_____@__________|
    070   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
    080   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
    090   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
    0A0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
    0B0   00 00 00 00 00 00 00 43 E0 00 00 08 0F FF 00 00   |_______C`________|
```

```
    0C0   00 0E 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
    0D0   02 00 0F 00 F8 3F 00 00 2D 15 F1 5F 40 00 00 00   |____x?__-_q_@___|
    0E0   00 00 04 00 01 00 00 00 04 00 20 51 00 00 00 00   |__________ Q____|
    0F0   00 00 00 00 80 7C 00 00 4C 64 6C 68 00 44 00 00   |_____|__Ldlh_D__|

0 hdmi .DEV
    000   00 01 00 98 20 00 00 00 00 00 00 00 04 8F 1F   |____ _____________|
    010   25 7D 02 7F 3F FF FF 00 00 00 80 18 00 07 00 01   |%}__?____________|
    020   00 00 00 01 00 00 00 00 00 00 00 02 00 02 00 00   |________________|
    030   00 00 3F FE 3F FE 00 00 00 00 00 00 02 00 39 63   |__?~?~________9c|
    040   00 40 00 0F 05 00 0F 00 00 00 00 00 44 00 3B 63   |_@___________D_;c|
    050   00 00 00 00 00 00 58 DA 01 A3 00 00 00 00 00 00   |______XZ_#______|
    060   0F 0F 00 00 00 00 00 00 00 00 00 00 A2 00 04 08   |____________"___|
    070   02 00 04 00 00 00 03 00 00 00 00 07 FF 00 00 00   |________________|
    080   00 00 00 FF 00 1F 00 70 38 0B 20 0F A3 04 1E 0F   |_______p8_ _#___|
    090   0B 20 0F 0B 20 0F 01 C0 FF A3 FF 03 00 02 00 00   |_ __ __@_#______|
    0A0   00 07 00 00 16 16 16 00 00 F0 00 F0 00 F0 00 00   |_________p_p_p__|
    0B0   00 00 00 00 00 00 00 00 00 00 02 00 00 03 C3 00   |______________C_|
    0C0   FF 01 01 01 01 01 01 01 01 01 01 01 01 00 00 00   |________________|
    0D0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
    0E0   00 00 00 00 00 01 00 00 00 01 00 00 00 00 00 82   |________________|
    0F0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|

0 cp .DEV
    000   00 00 00 00 00 00 00 00 00 02 00 40 00 00 00 00   |___________@____|
    010   08 00 00 00 00 01 00 00 00 00 00 20 00 00 00 00   |___________ ____|
    020   20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   | _______________|
    030   00 00 00 00 00 29 00 00 00 00 80 80 00 00 00 00   |_____)__________|
    040   5C 02 3B D4 03 33 00 00 00 00 00 00 00 00 00 00   |\_;T_3__________|
    050   C0 00 40 00 00 00 00 08 00 00 00 00 01 00 00 00   |@_@_____________|
    060   00 00 20 00 00 00 00 00 F0 04 00 00 00 00 00 00   |__ _____p_______|
    070   00 00 00 10 04 01 00 FF FF FF FF 05 C0 00 00 00   |___________@____|
    080   00 C0 04 00 0C 03 0B 00 00 00 20 40 00 00 00 40   |_@_________ @___@|
    090   00 40 00 00 00 00 00 F0 00 00 00 00 00 00 00 00   |_@_____p________|
    0A0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 60 00   |______________`_|
    0B0   00 3F FF 00 00 F9 08 29 00 00 03 00 00 18 00 12   |_?___y_)________|
    0C0   00 00 00 39 44 91 00 00 00 2D 00 60 1A 00 80 00   |___9D____-_`____|
    0D0   00 00 00 00 00 00 00 00 00 00 00 19 64 12 C5 78   |____________d_Ex|
    0E0   00 20 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |_ _____________|
    0F0   00 40 04 D4 00 00 00 15 03 33 80 00 04 65 00 78   |_@_T_____3___e_x|

0 ksv .DEV
    000   FF FF FF FF FF 00 00 00 00 00 00 00 00 00 00 00   |________________|
    010   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
    020   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
    030   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
    040   83 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
    050   00 00 10 00 00 00 10 00 00 00 10 00 00 00 10 00   |________________|
    060   00 DC 00 DC 00 DC 00 DC 00 DC 00 DC 00 DC 00 DC   |_\_\_\_\_\_\_\_\|
    070   00 00 01 00 01 00 01 00 00 88 04 00 00 00 00 14   |________________|
    080   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
    090   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
    0A0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
    0B0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
    0C0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
```

```
    0D0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
    0E0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
    0F0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|

0 dpll .DEV
    000   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
    010   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
    020   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
    030   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
    040   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
    050   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
    060   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
    070   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
    080   80 20 80 02 02 0A 0A 0A F8 66 FC 13 F8 15 DE 86   |_ ______xf|_x_^_|
    090   C1 91 82 70 00 00 00 00 FF 0F FF 00 00 00 00 00   |A__p____________|
    0A0   00 02 02 FF FF FF F8 66 FC 13 F8 15 DE 86 C1 91   |______xf|_x_^_A_|
    0B0   82 70 FF 0F FF 01 00 00 00 00 00 00 6F 02 FA      |_p____________o_z|
    0C0   C0 98 80 00 30 02 03 00 00 00 00 00 40 0A 96 00   |@___0_______@___|
    0D0   25 23 E1 AE FF E1 05 00 00 1B 04 00 00 00 00 00   |%#a._a__________|
    0E0   00 00 00 00 06 06 00 00 00 00 00 00 00 00 00 00   |________________|
    0F0   00 00 00 00 00 00 00 00 00 0D F8 0F E1 00 00 00   |__________x_a___|

0 cec .DEV
    000   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
    010   00 00 13 57 00 00 00 00 00 00 00 00 00 00 00 00   |___W____________|
    020   00 00 00 00 00 00 00 10 FF 0F 3E 07 00 0F BB 0E   |__________>___;_|
    030   AE 10 C7 0C EF 0B E2 0D FB 08 64 06 D1 09 F6 02   |._G_o_b_{_d_Q_v_|
    040   19 05 3E 06 4B 03 AC 0C 96 02 00 E0 00 01 0C 03   |__>_K_,____`____|
    050   25 04 32 00 00 00 00 00 00 00 00 00 00 00 00 00   |%_2_____________|
    060   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
    070   00 00 00 00 00 00 00 00 6D 8F 82 04 0D 70 42 41   |________m____pBA|
    080   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
    090   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
    0A0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
    0B0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
    0C0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
    0D0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
    0E0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
    0F0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|

0 info .DEV
    000   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
    010   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
    020   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
    030   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
    040   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
    050   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
    060   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
    070   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
    080   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
    090   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
    0A0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
    0B0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
    0C0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
    0D0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
```

```
    0E0   82 00 00 84 00 00 83 00 00 85 00 00 81 00 00 04   |________________|
    0F0   00 00 05 00 00 06 00 00 0A 00 00 00 00 00 00 00   |________________|

0 edid .DEV
    000   00 FF FF FF FF FF FF 00 1C 23 00 42 00 00 00 00   |_________#_B____|
    010   2D 19 01 03 81 00 00 96 0F EE 91 A3 54 4C 99 26   |-________n_#TL_&|
    020   0F 50 54 00 00 00 D1 C0 01 01 01 01 01 01 01 01   |_PT___Q@________|
    030   01 01 01 01 01 01 02 3A 80 18 71 38 2D 40 58 2C   |_______:__q8-@X,|
    040   45 00 00 00 00 00 00 1E 00 00 00 FF 00 47 41 43   |E____________GAC|
    050   59 53 70 72 6F 74 6F 30 30 31 00 00 00 FD 00 3B   |YSproto001___}_;|
    060   3D 43 44 95 00 0A 20 20 20 20 20 20 00 00 00 FC   |=CD___      ___||
    070   00 53 75 6E 72 69 73 65 20 4D 65 67 61 31 01 2A   |_Sunrise Mega1_*|
    080   02 03 12 01 6B 00 0C 03 10 00 00 21 80 00 FF 00   |____k______!____|
    090   41 90 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |A_______________|
    0A0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
    0B0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
    0C0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
    0D0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
    0E0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |________________|
    0F0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ED   |_______________m|

Plug into bat, which recognizes us:

DELTA
dpll/BE 02 03  hdmi/04 20 23  hdmi/05 00 30  hdmi/06 00 94  hdmi/07 00 A7
hdmi/08 00 80  hdmi/09 00 04  hdmi/0A 00 38  hdmi/0B 00 04  hdmi/0C 00 38
hdmi/1C 00 0B  hdmi/1E 00 08  hdmi/1F 01 98  hdmi/21 00 58  hdmi/23 01 2C
hdmi/25 00 94  hdmi/26 00 08  hdmi/27 00 CA  hdmi/28 00 08  hdmi/29 00 CA
hdmi/2B 02 08  hdmi/2D 02 08  hdmi/2F 00 0A  hdmi/31 00 0A  hdmi/32 3F 00
hdmi/33 FE 48  hdmi/34 3F 00  hdmi/35 FE 48  hdmi/51 00 4A  hdmi/52 00 3E
hdmi/D2 00 02  hdmi/D6 00 09  hdmi/E0 00 4A  hdmi/E1 00 3E  hdmi/FF 00 11
cp/A3 00 04  cp/A4 00 64  cp/B1 3F 8D  cp/B2 FF 39  cp/B3 00 28
cp/B8 00 07  cp/B9 00 48  cp/CD 00 FF  cp/CE 80 8F  cp/E0 00 50
cp/FA 80 00  cp/FF 78 48  io/12 81 C0  io/21 00 08  io/42 08 14
io/5B 04 0A  io/6A 00 53  io/6F 00 01  io/83 00 42  io/D2 0F 9F
io/D3 00 01  io/D4 F8 F1  io/D5 1F 2F  ok
```

- Plug into an HDMI source and verify understood
- Verify valid video timing coming out
- ***Send code to James with instructions to deliver good video to him in pixel bus.***

4.1.4 Video Section

- Power all the VX and VC buses from external supply (1.8 and 3.3V)
- Verify JTAG lines work electrically.
- Learn how to prove JTAG works logically.
- Design, code and test JTAG support.
- Get daisy chained JTAG streams from James and test.

- Verify I^2C functions if implemented.
- Examine pixel bus signal integrity

4.2 Module PCB

4.2.1 Power Section

- Check unstuffed board for shorts.
- Check stuffed board for shorts.
- Apply external power (1.8V and 3.3V)

4.2.2 GA144 Section

- Configure Sparkfun FTDI chip from PC using procedure like EVB.
- Set jumpers J108 in, J101 in, J403 1-2, J502 1-2, J501 out.
- Plug into PCB and power up (insert jumpers in J201, J202, J203 to enable).
- Verify control of USB reset at J108 and J502.3.
- Test basic IDE operations
- Run selftest on GA144.
- Verify ability to program and boot from flash with a simple piece of code.
- Verify new flash with Charley code.
- Test watchdog circuit.
- Check the testpoints on computer section.
- Get crystal osc going for module and control channels.

4.2.3 Module communications

- Test circuitry to talk Dean's protocol
- Verify clean operation with module, using test patterns

4.2.4 Control Channel

- Connect to control PCB and verify signals
- Test serpentine reset circuit
- Design, code and test control protocol

4.2.5 Pixel Bus

- Connect to working control PCB
- Devise test with known video (pF on PC)
- Verify video formats, indexing and so on
- Check signal integrity

4.3 Production Software

- Instrument a module PCB for power measurement
- Floor planning of module PCBs
- Finalize design of pixel pipeline
- Work up basic path to module and test
- Work up control protocols to module
- Design, code and test watchdog management
- Optimize for robustness
- Optimize for min power

4.4 Production Hardware

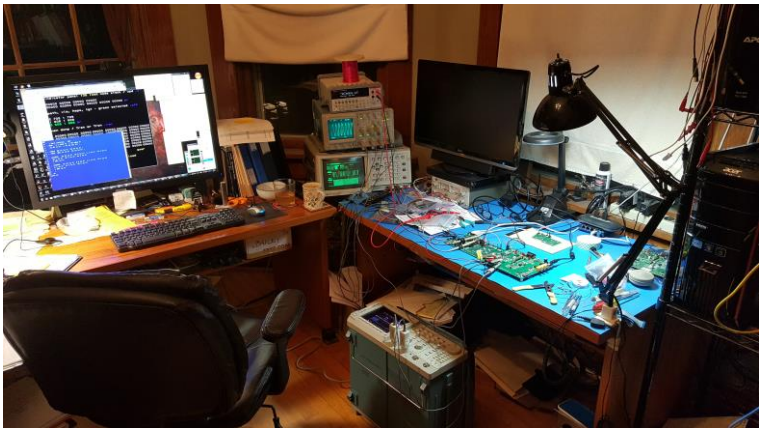
- Apply lessons learned
- Remove needless test / evaluation circuitry
- Define mods to support production test/commissioning
- Apply to drawings

5. Hardware and Schematics

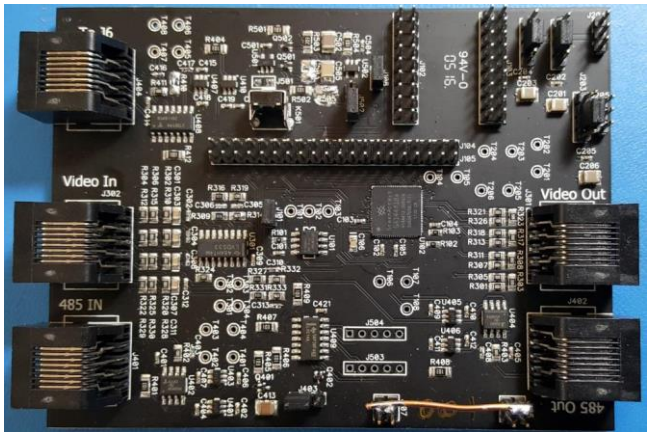
This section shows selected photos during prototype PCB and sign set-up, and schematics of the prototype PCBs.

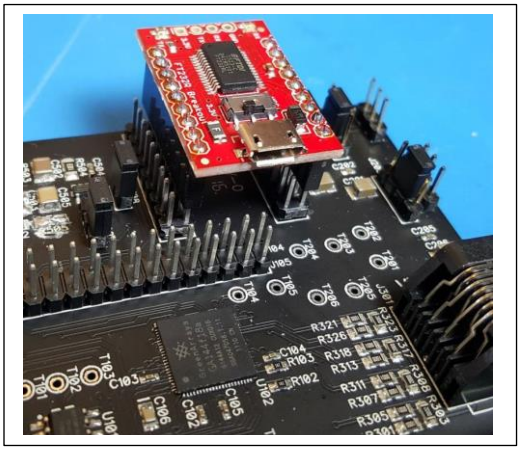
5.1 Photos

Here is the bench set-up we used for initial testing of the ADV7611 HDMI receiver chip. An EVB001 board was used to drive the I2C lines and scopes were used to observe the pixel bus data, clock and framing lines.

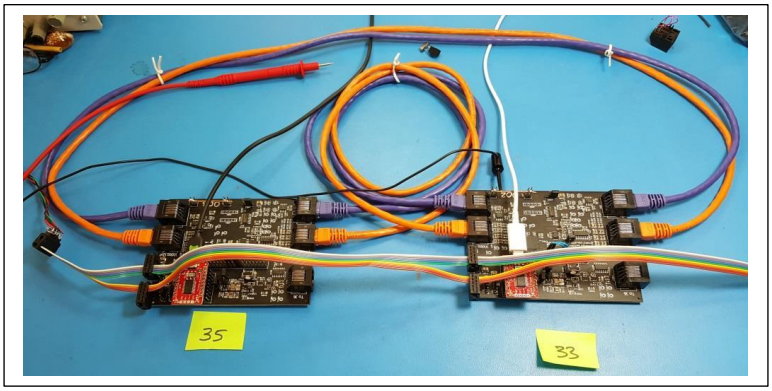


The first board built was the VMOD prototype, set up almost as a breadboard for testing and evaluation. The stuffed board is shown on the left and the SparkFun FTDI dongle is on the right.



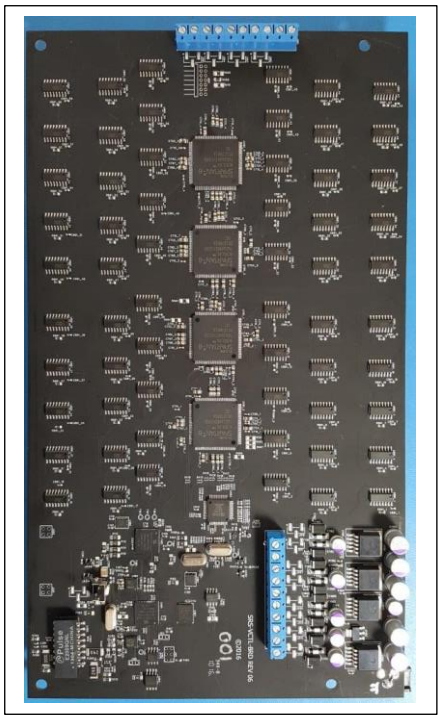


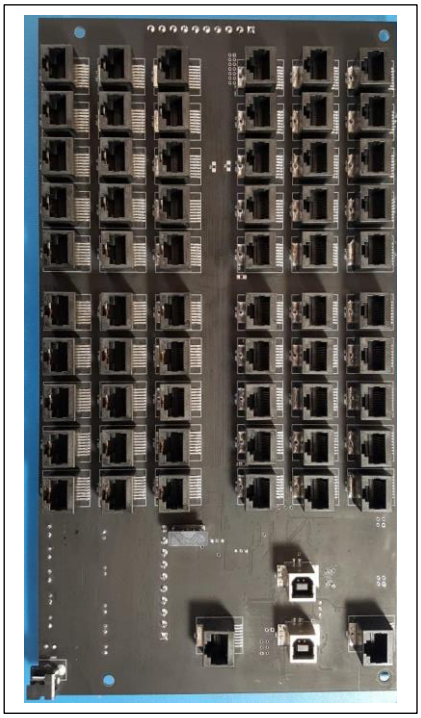
Small scale testing of pixel pipeline and sign management daisy chain were done using two and three boards on a simple bench as shown at right. Initial signals were produced by an EVB001 generating fake video.



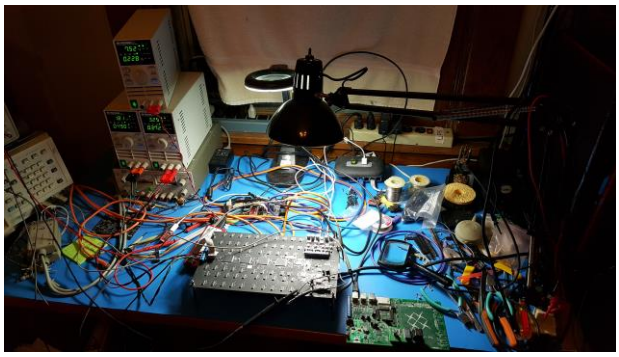
The VCTL boards finally arrived in March 2016, two months later than we'd hoped, allowing us to stop working with fake video from an EVB001. Here are top and bottom views of this board.

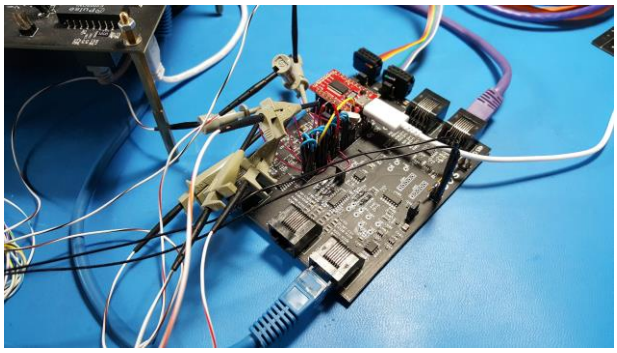
Very little rework was required. We had to add a series RC from LLC to ground (22pF and 49.9Ω) to terminate that line; its signal quality was poor, with much ringing and crosstalk, resulting in unreliable pixel clocking. After fixing this the parallel pixel bus was well behaved into the FPGAs.





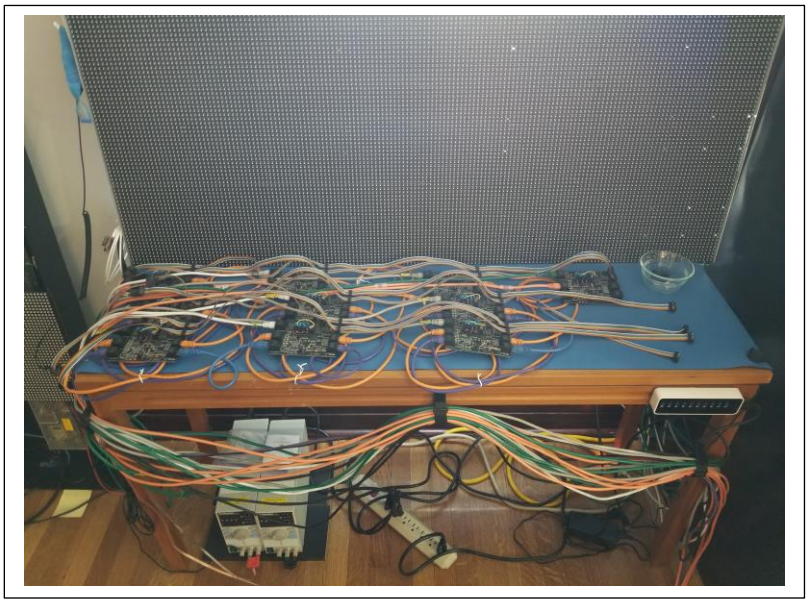
Here are bench set-ups used for VCTL debugging and for critical examination of the pixel pipeline.



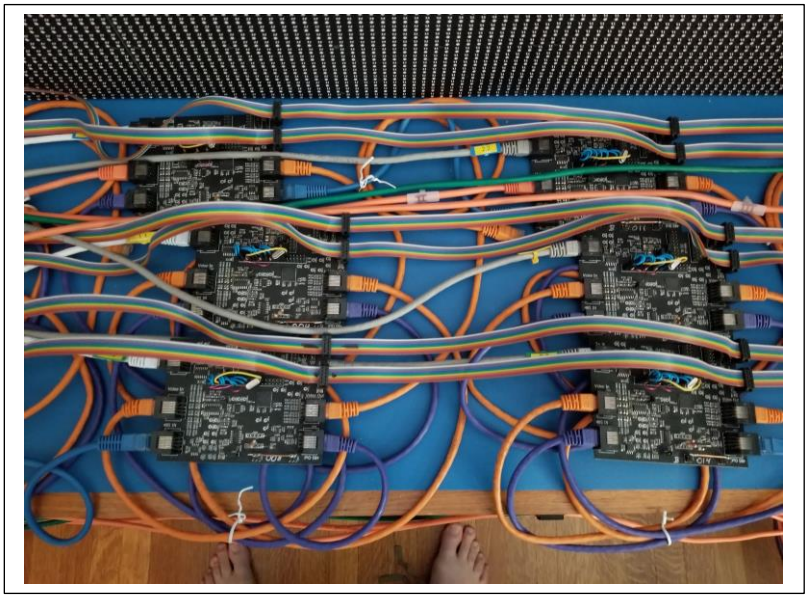


Here is the very small (4x3 module) subset of a display under construction and the first live video from a Windows notebook. Moire is an artifact of image capture, view at large size for better appearance.

The final working prototype was configured at 3x3 modules with one additional module in the upper right corner. Here is the sign bench before it was dismantled. Pixel pipeline is carried on the orange Cat5e cables; the sign management daisy chain uses purple cables with blue making the hops between columns. The cables connecting each VMOD with its Sunrise LED module leave the left side of the VMOD and loop around the left end of the bench, coming across the front in the cable bundle and proceeding into the module trays on the right. In production the VMODs would be much smaller and would probably become integral parts of the cable harness.



Detail of the VMOD array's wiring is here.

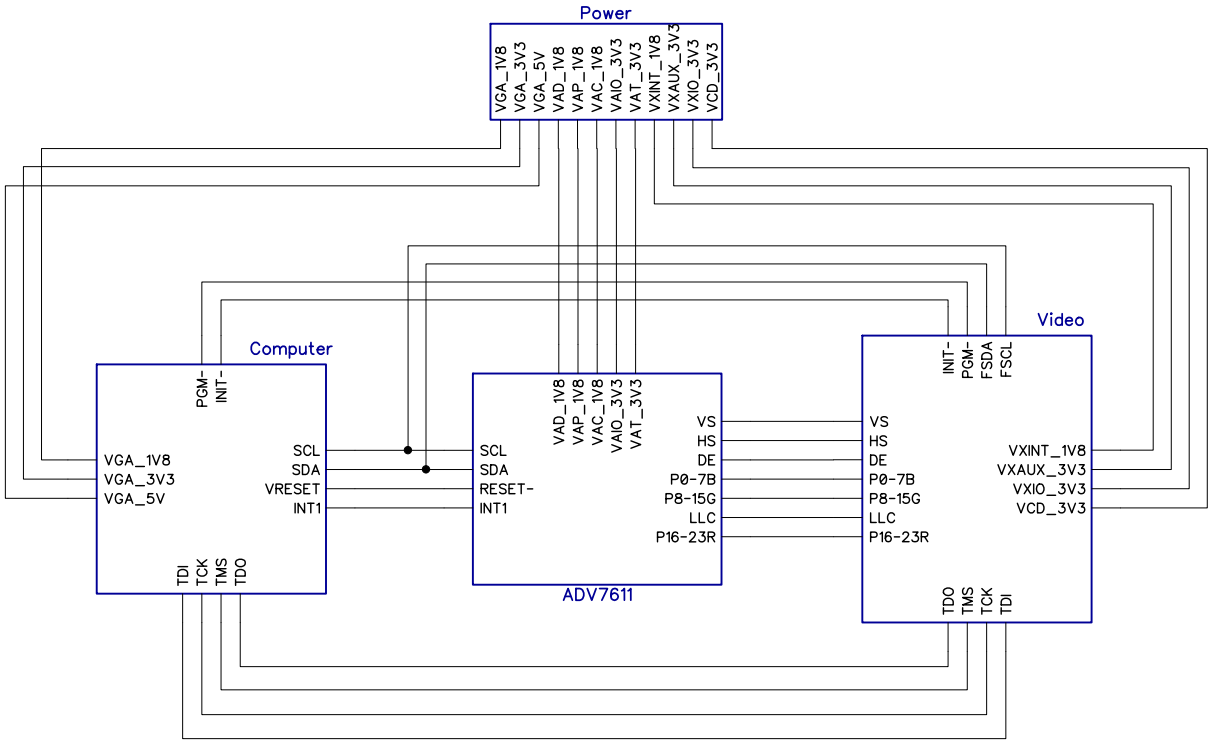


5.2 Schematics

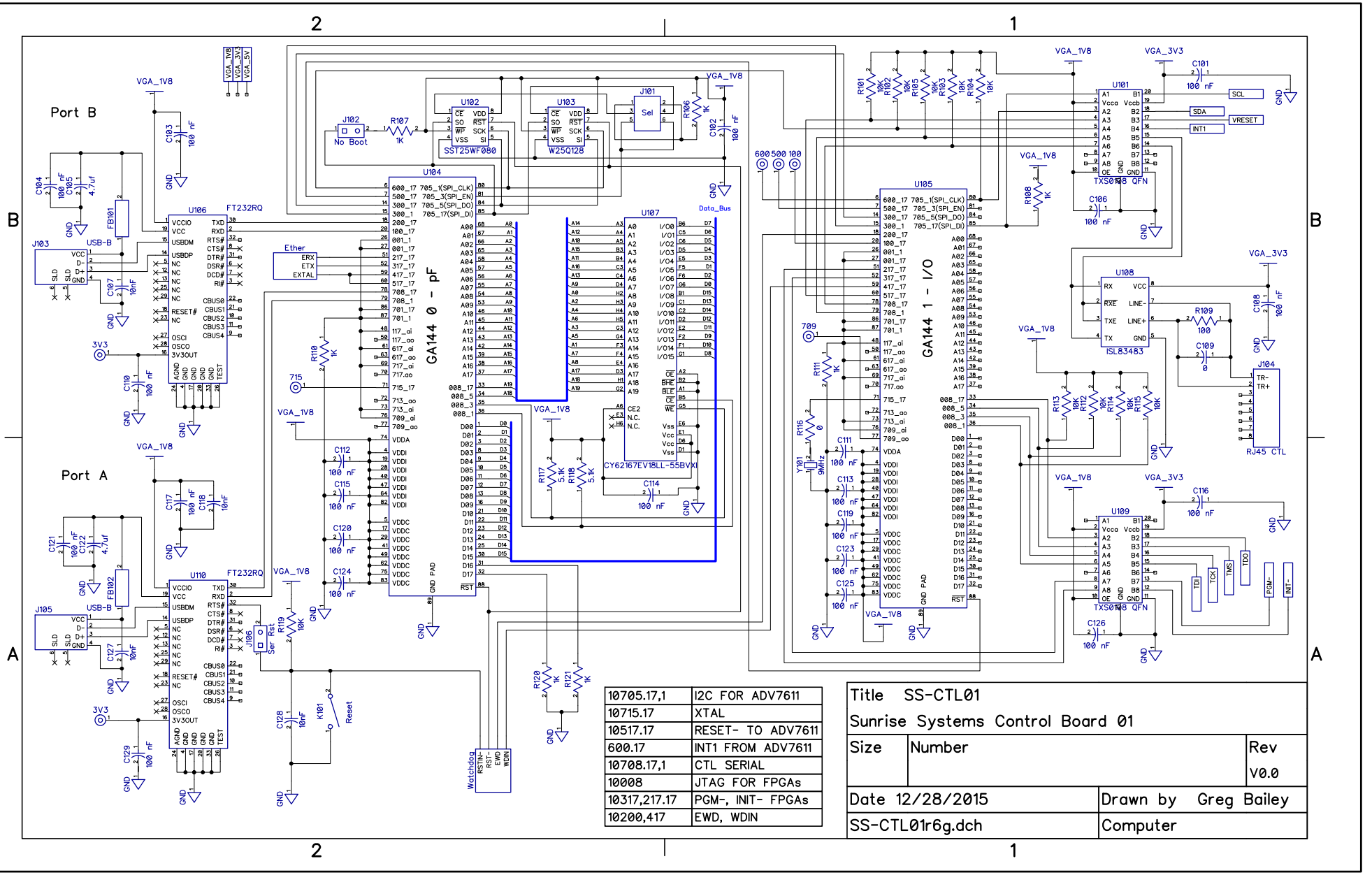
The following pages are schematic drawings for the PCBs used in this project.

The first 12 pages are the VCTL PCB discussed above. Small changes occurred before manufacture and a small amount of rework was done for testing in the prototype.

The next 6 pages are the VMOD PCB modified for production. Headers, test points, optioning jumpers and so on have been removed.

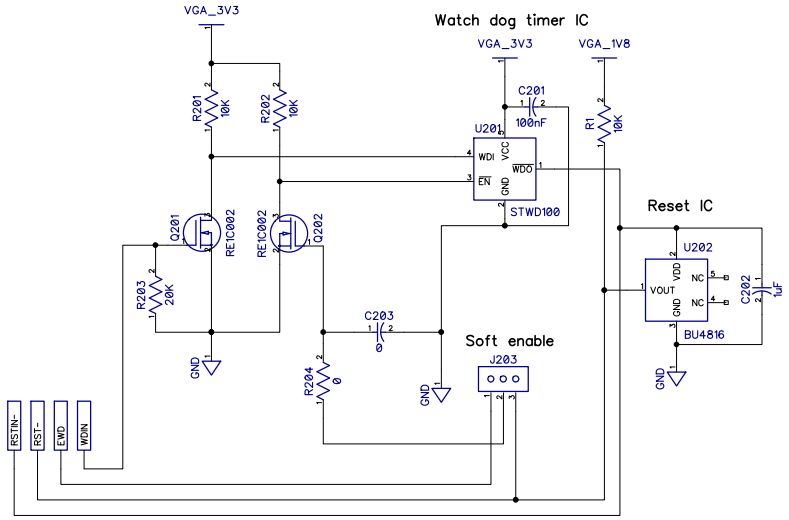


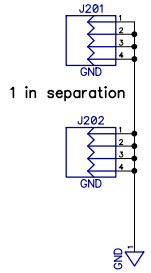
Title SS-CTL01 Sunrise Systems Control Board 01		
Size	Number	Rev V0.0
Date 12/28/2015	Drawn by Greg Bailey	
SS-CTL01r6g.dch	Top	



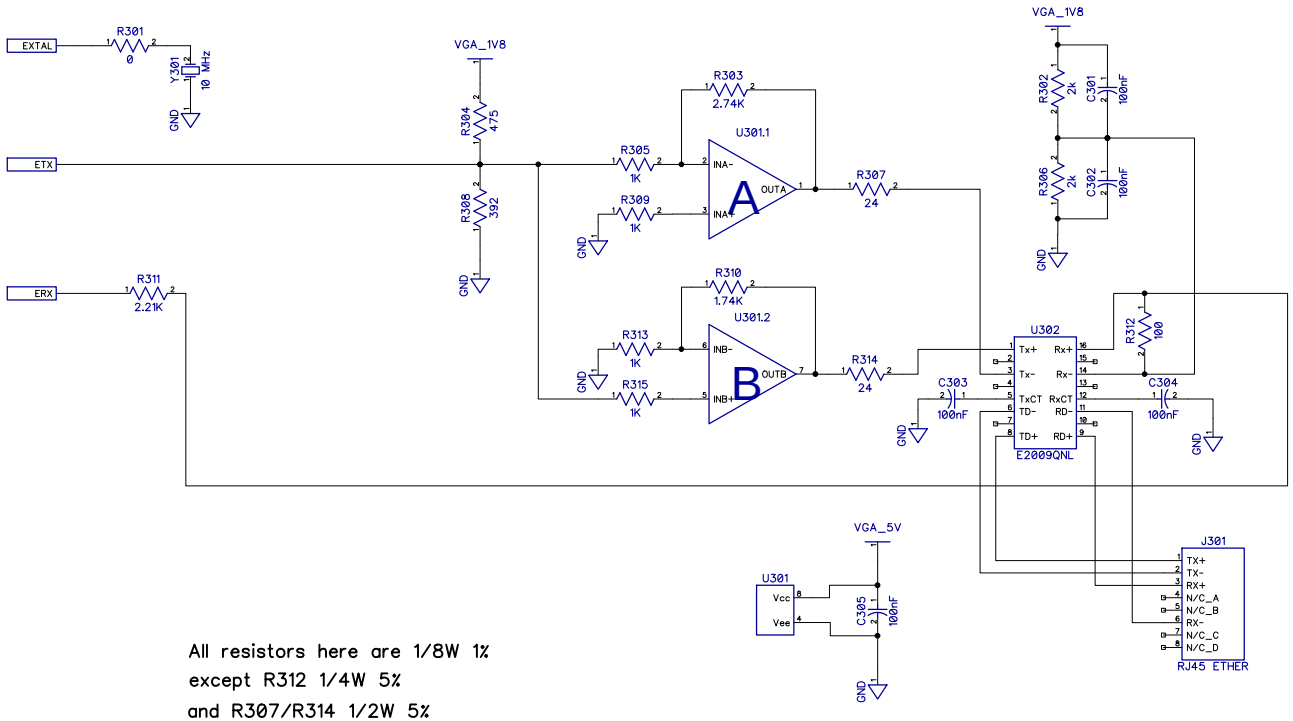
10705.17,1	I2C FOR ADV7611
10715.17	XTAL
10517.17	RESET- TO ADV7611
600.17	INT1 FROM ADV7611
10708.17,1	CTL SERIAL
10008	JTAG FOR FPGAs
10317,217.17	PGM-, INIT- FPGAs
10200,417	EWD, WDIN

Title SS-CTL01		
Sunrise Systems Control Board 01		
Size	Number	Rev V0.0
Date 12/28/2015	Drawn by Greg Bailey	
SS-CTL01r6g.dch	Computer	



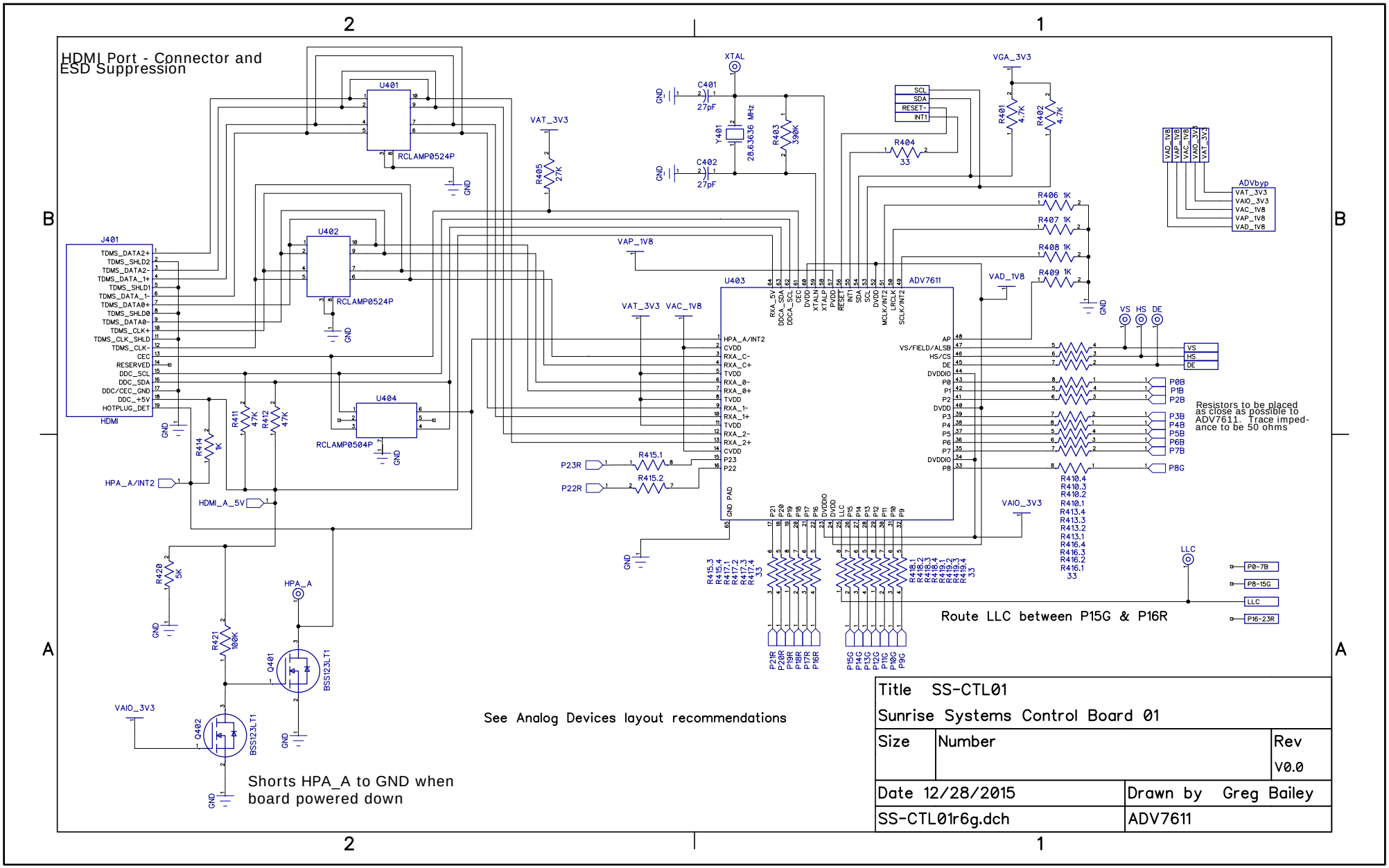


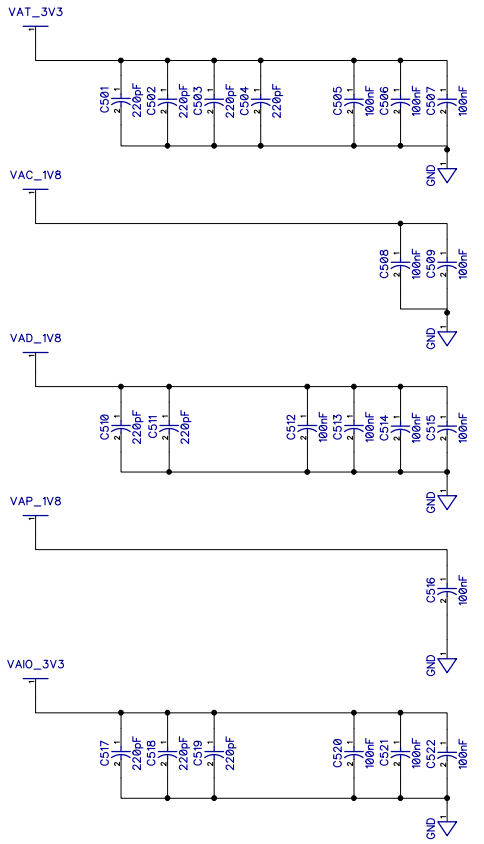
Title SS-CTL01 Sunrise Systems Control Board 01		
Size	Number	Rev V0.0
Date 12/28/2015	Drawn by Greg Bailey	
SS-CTL01r6g.dch	Watchdog	



All resistors here are 1/8W 1%
except R312 1/4W 5%
and R307/R314 1/2W 5%

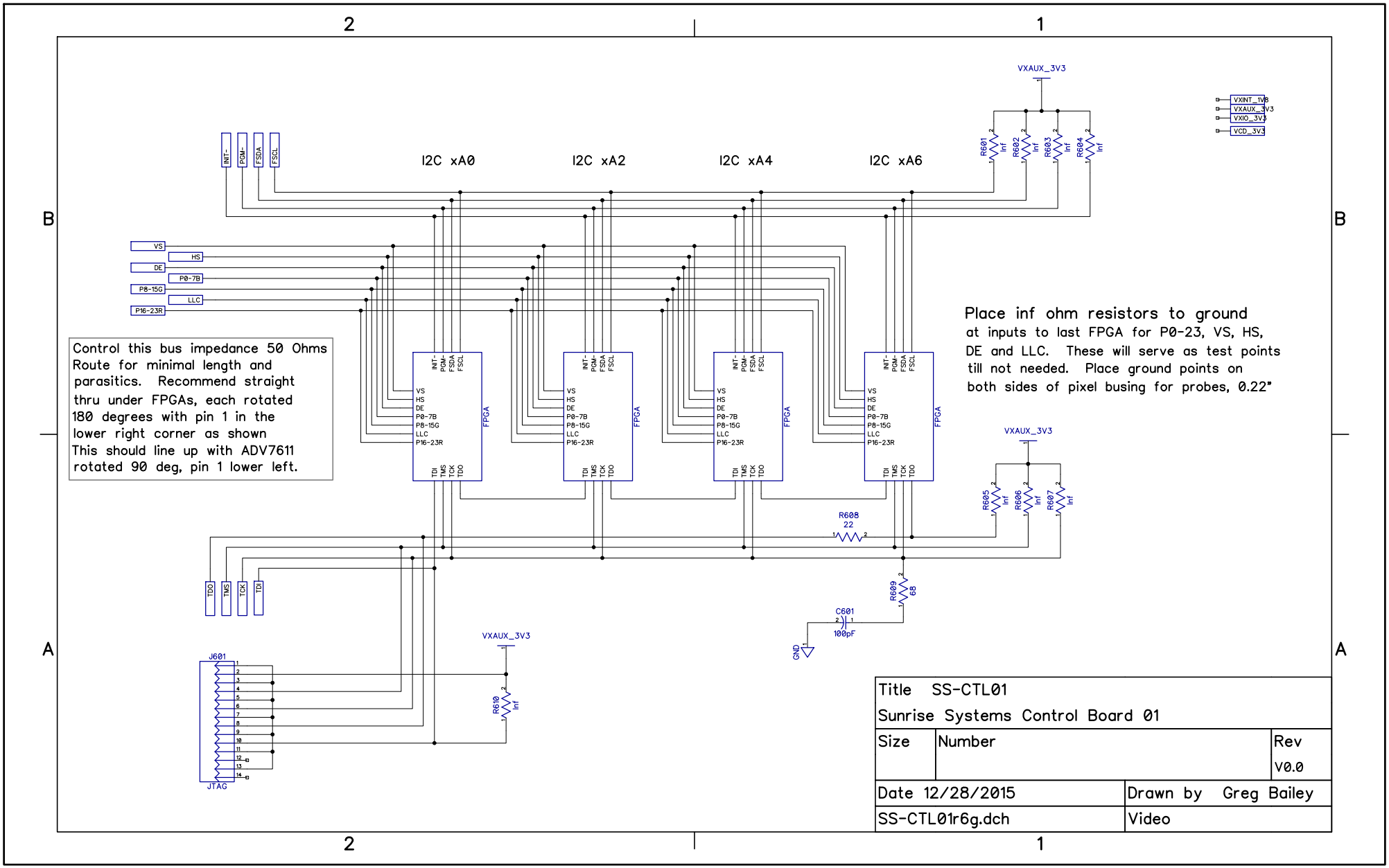
Title SS-CTL01		
Sunrise Systems Control Board 01		
Size	Number	Rev V0.0
Date 12/28/2015	Drawn by Greg Bailey	
SS-CTL01r6g.dch	Ether	

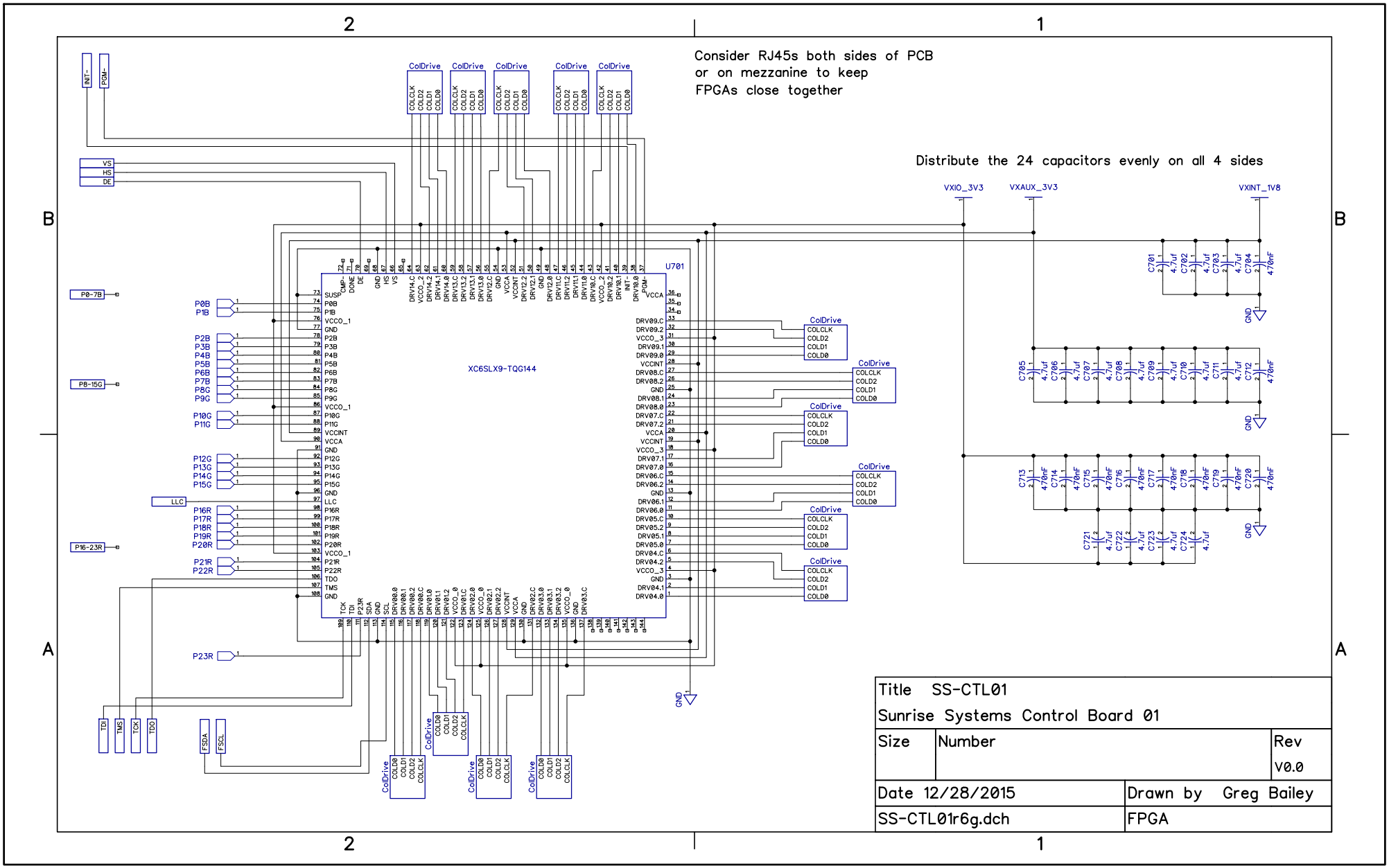

HDMI Port - Connector and ESD Suppression
J401
HDMI
U401
U402
U403
U404
ADV7611
RCLAMP0524P
RCLAMP0504P
XTAL
C401
C402
27pF
Y401
28.63636 MHz
R401 4.7K
R402 4.7K
R403 390K
R404 33
R405 27K
R406 1K
R407 1K
R408 1K
R409 1K
R410.4
R410.3
R410.2
R410.1
R413.4
R413.3
R413.2
R413.1
R416.4
R416.3
R416.2
R416.1
R411 4.7K
R412 4.7K
R414 1K
R420 5K
R421 100K
R415.1
R415.2
R415.3
R415.4
R417.1
R417.2
R417.3
R417.4
R418.1
R418.2
R418.3
R418.4
R419.1
R419.2
R419.3
R419.4
33
VGA_3V3
VAT_3V3
VAP_1V8
VAD_1V8
VAC_1V8
VAIO_3V3
GND
SCL
SDA
RESET-
INT1
VS
HS
DE
ADVbyp
VAT_3V3
VAIO_3V3
VAC_1V8
VAP_1V8
VAD_1V8
P0B
P1B
P2B
P3B
P4B
P5B
P6B
P7B
P8G
P23R
P22R
HPA_A/INT2
HDMI_A_5V
HPA_A
Q401
Q402
BSS123LT1
LLC
P0-7B
P8-15G
P16-23R
Resistors to be placed as close as possible to ADV7611. Trace impedance to be 50 ohms
Route LLC between P15G & P16R
See Analog Devices layout recommendations
Shorts HPA_A to GND when board powered down
Title SS-CTL01
Sunrise Systems Control Board 01
Size
Number
Rev V0.0
Date 12/28/2015
Drawn by Greg Bailey
SS-CTL01r6g.dch
ADV7611

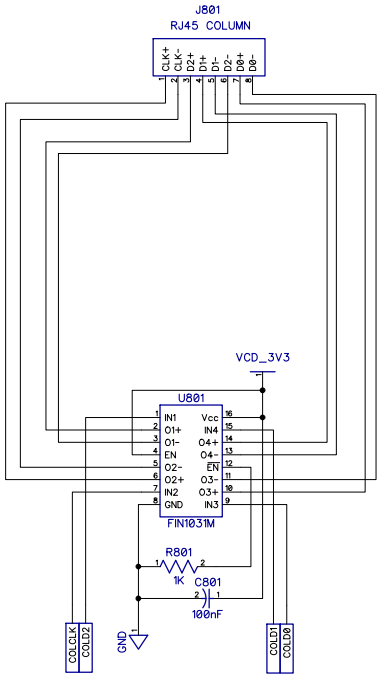


See Analog Devices layout recommendations

Title SS-CTL01 Sunrise Systems Control Board 01		
Size	Number	Rev V0.0
Date 12/28/2015	Drawn by Greg Bailey	
SS-CTL01r6g.dch	ADVbyp	

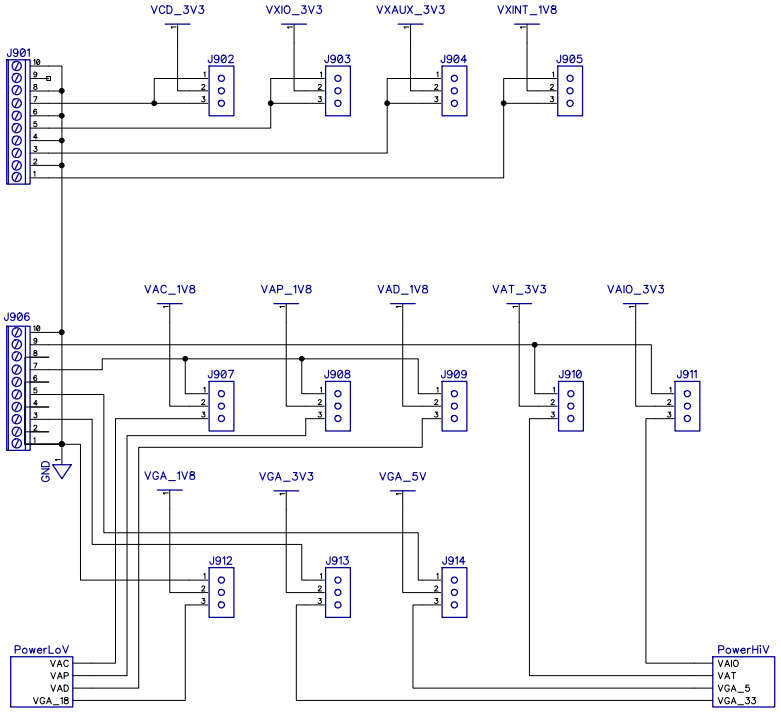

2
1
B
A
VXAUX_3V3
VXINT_1V8
VXAUX_3V3
VXIO_3V3
VCD_3V3
INIT-
PGM-
FSDA
FSCL
I2C xA0
I2C xA2
I2C xA4
I2C xA6
R601 Inf
R602 Inf
R603 Inf
R604 Inf
VS
HS
DE
P0-7B
P8-15G
LLC
P16-23R
Control this bus impedance 50 Ohms
Route for minimal length and
parasitics. Recommend straight
thru under FPGAs, each rotated
180 degrees with pin 1 in the
lower right corner as shown
This should line up with ADV7611
rotated 90 deg, pin 1 lower left.
FPGA
TDI
TMS
TCK
TDO
Place inf ohm resistors to ground
at inputs to last FPGA for P0-23, VS, HS,
DE and LLC. These will serve as test points
till not needed. Place ground points on
both sides of pixel busing for probes, 0.22"
R605 Inf
R606 Inf
R607 Inf
R608 22
R609 68
C601 100pF
GND
TDO
TMS
TCK
TDI
J601
JTAG
R610 Inf
Title SS-CTL01
Sunrise Systems Control Board 01
Size
Number
Rev V0.0
Date 12/28/2015
Drawn by Greg Bailey
SS-CTL01r6g.dch
Video


Consider RJ45s both sides of PCB
or on mezzanine to keep
FPGAs close together
Distribute the 24 capacitors evenly on all 4 sides
U701
XC6SLX9-TQG144
ColDrive
VXIO_3V3
VXAUX_3V3
VXINT_1V8
GND
Title SS-CTL01
Sunrise Systems Control Board 01
Size
Number
Rev
V0.0
Date 12/28/2015
Drawn by Greg Bailey
SS-CTL01r6g.dch
FPGA

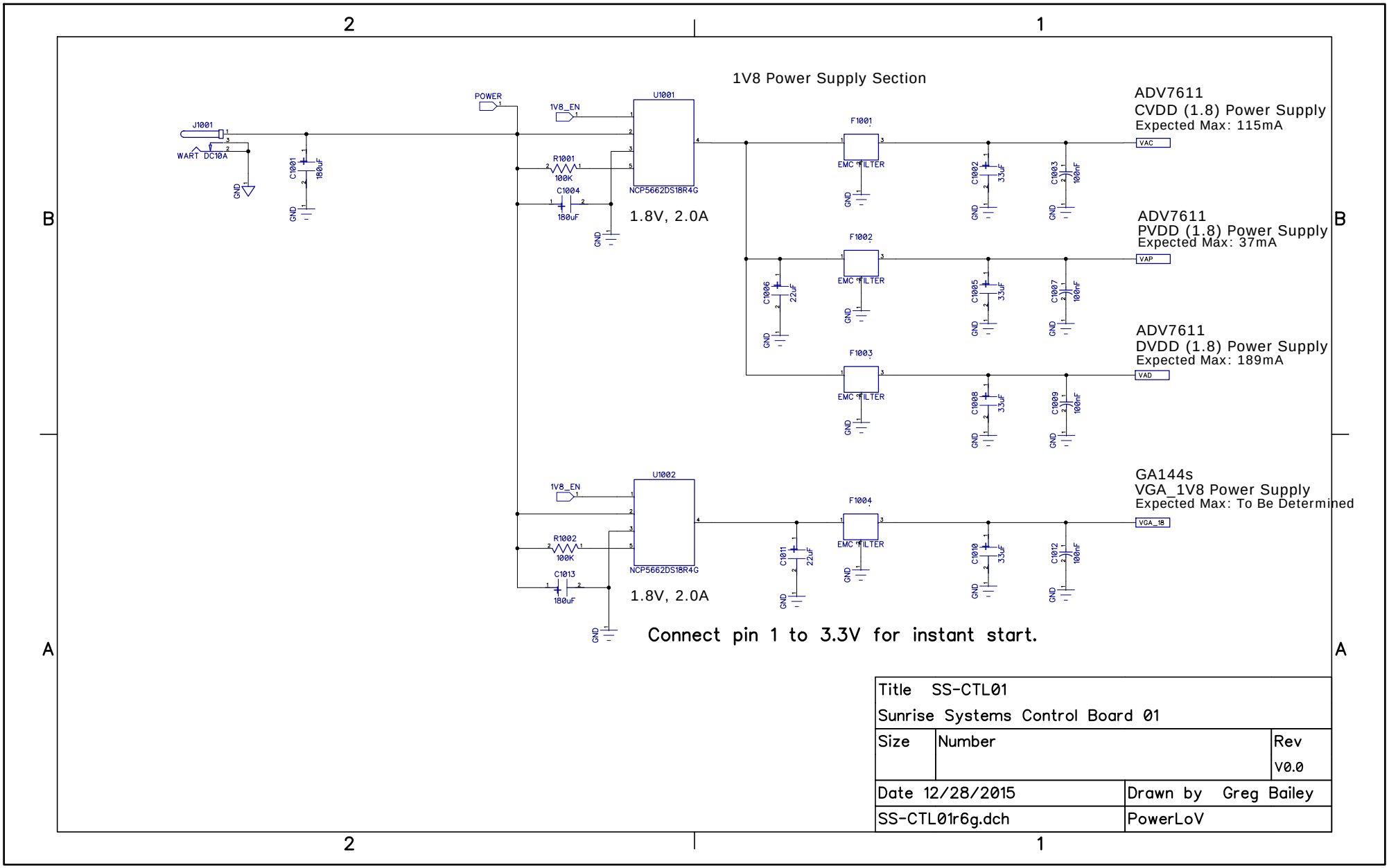


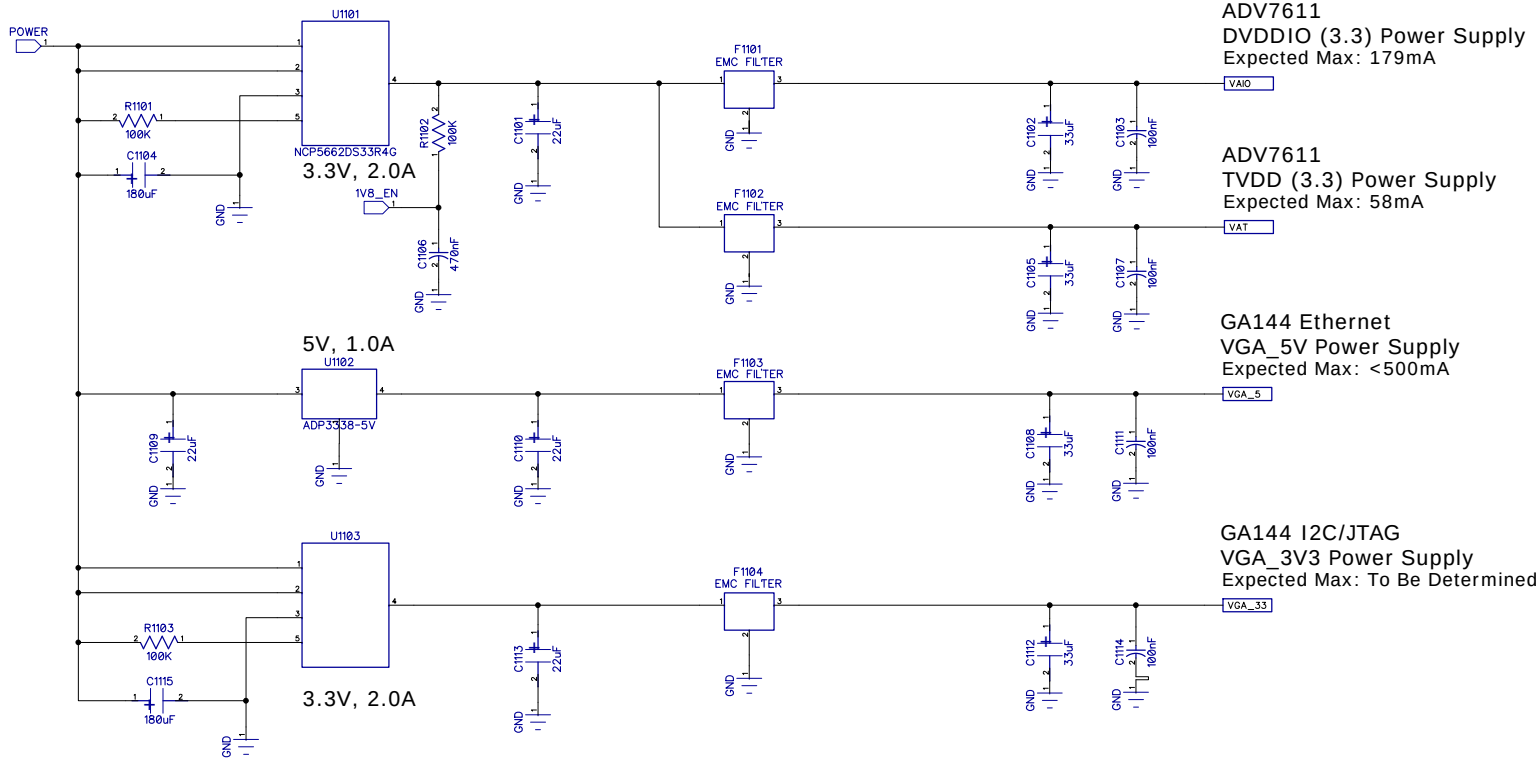
Title SS-CTL01 Sunrise Systems Control Board 01		
Size	Number	Rev V0.0
Date 12/28/2015	Drawn by Greg Bailey	
SS-CTL01r6g.dch	ColDrive	

External Power



Title SS-CTL01		
Sunrise Systems Control Board 01		
Size	Number	Rev V0.0
Date 12/28/2015	Drawn by Greg Bailey	
SS-CTL01r6g.dch	Power	


1V8 Power Supply Section
POWER
J1001
WART DC10A
C1001
180uF
GND
1V8_EN
U1001
NCP5662DS18R4G
R1001
100K
C1004
180uF
1.8V, 2.0A
F1001
EMC FILTER
C1002
33uF
C1003
100nF
VAC
ADV7611
CVDD (1.8) Power Supply
Expected Max: 115mA
F1002
C1006
22uF
C1005
33uF
C1007
100nF
VAP
ADV7611
PVDD (1.8) Power Supply
Expected Max: 37mA
F1003
C1008
33uF
C1009
100nF
VAD
ADV7611
DVDD (1.8) Power Supply
Expected Max: 189mA
U1002
R1002
100K
C1013
180uF
1.8V, 2.0A
F1004
C1011
22uF
C1010
33uF
C1012
100nF
VGA_18
GA144s
VGA_1V8 Power Supply
Expected Max: To Be Determined
Connect pin 1 to 3.3V for instant start.
Title SS-CTL01
Sunrise Systems Control Board 01
Size
Number
Rev
V0.0
Date 12/28/2015
Drawn by Greg Bailey
SS-CTL01r6g.dch
PowerLoV



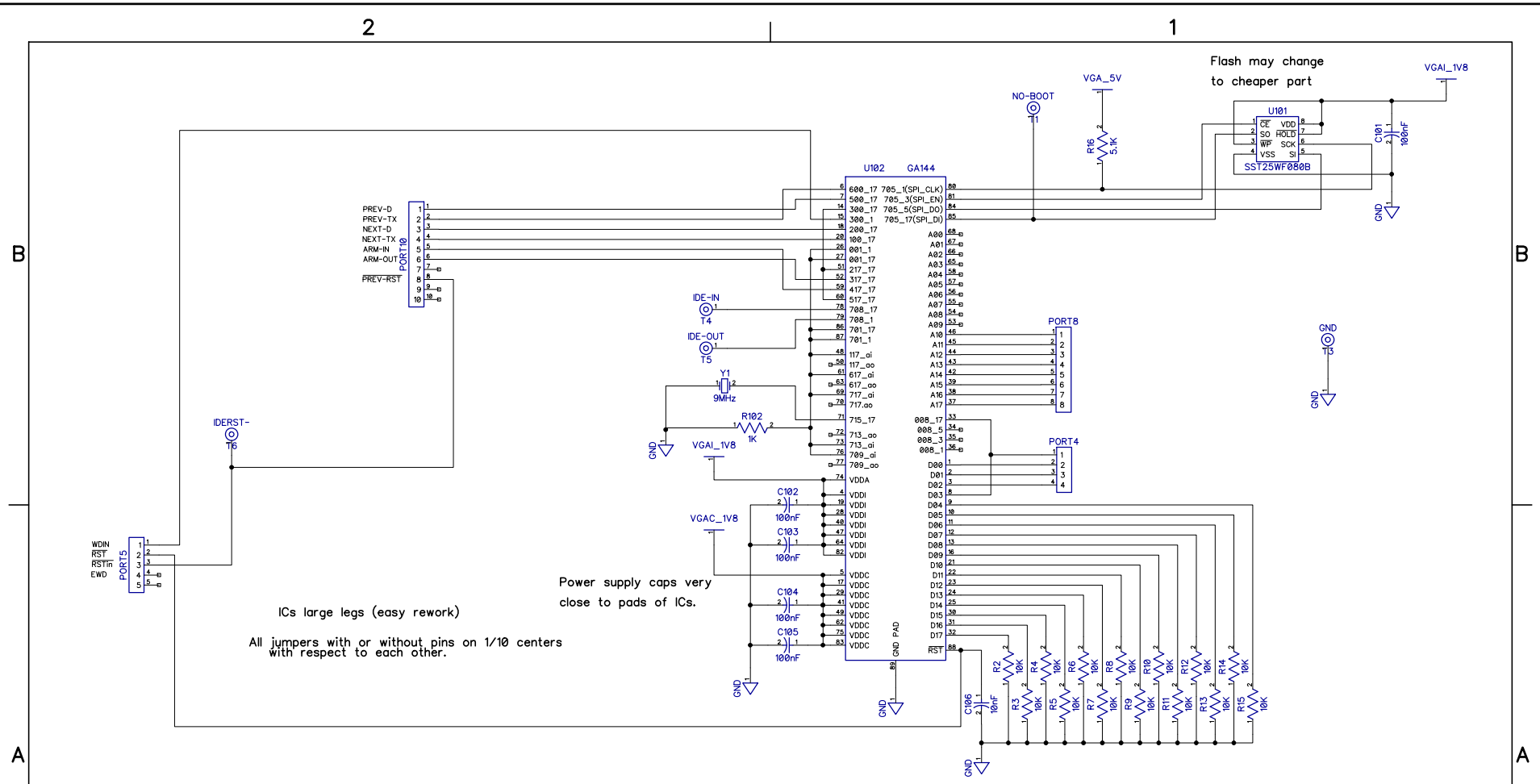
Title SS-CTL01		
Sunrise Systems Control Board 01		
Size	Number	Rev V0.0
Date 12/28/2015	Drawn by Greg Bailey	
SS-CTL01r6g.dch	PowerHiV	

Check pin labeling and patterns against data sheets

When a testpoint has a ground right beside it, put holes on 0.22" centers to fit active probe
In prototype build, all parts should be 5% precision or better

Resistance 0 means zero ohm resistor.
Cap 0 means don't populate but don't mask pads.

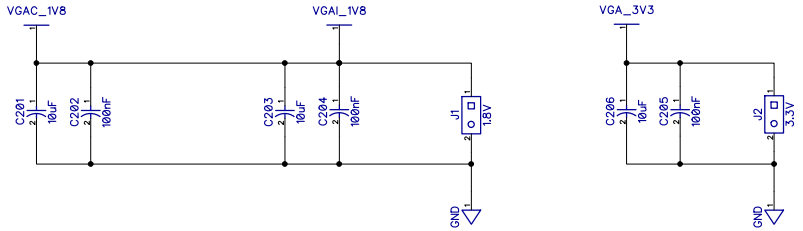
Title SS-MOD01 Production Sunrise Systems Module Video Distrib Board 01		
Size	Number	Rev V0.0
Date 06/03/2016	Drawn by Greg Bailey	
SS-MOD01r8.dch	Top	



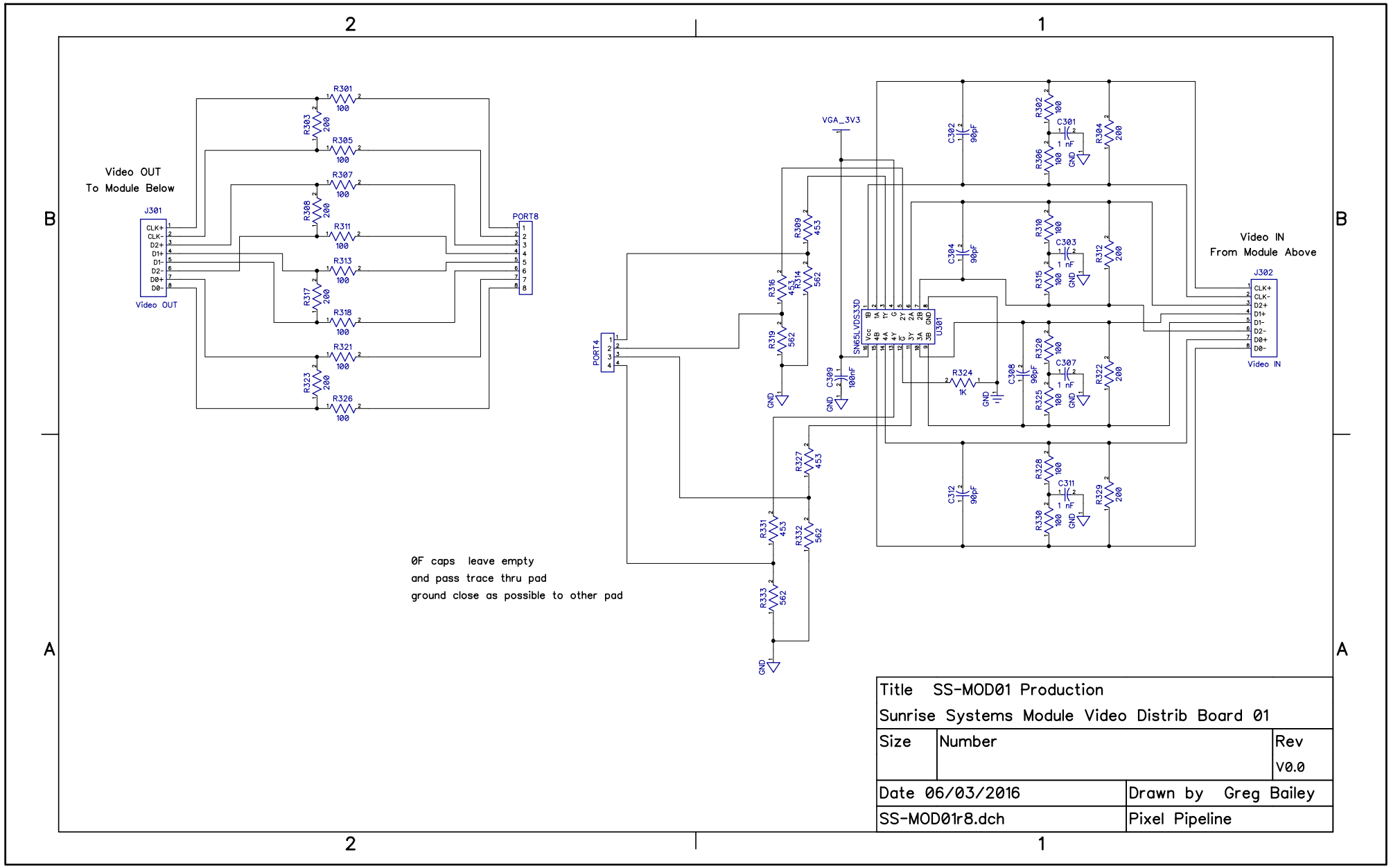
DipTrace doesn't haved the needed ICs, Correct ICs/Patterns must be used.

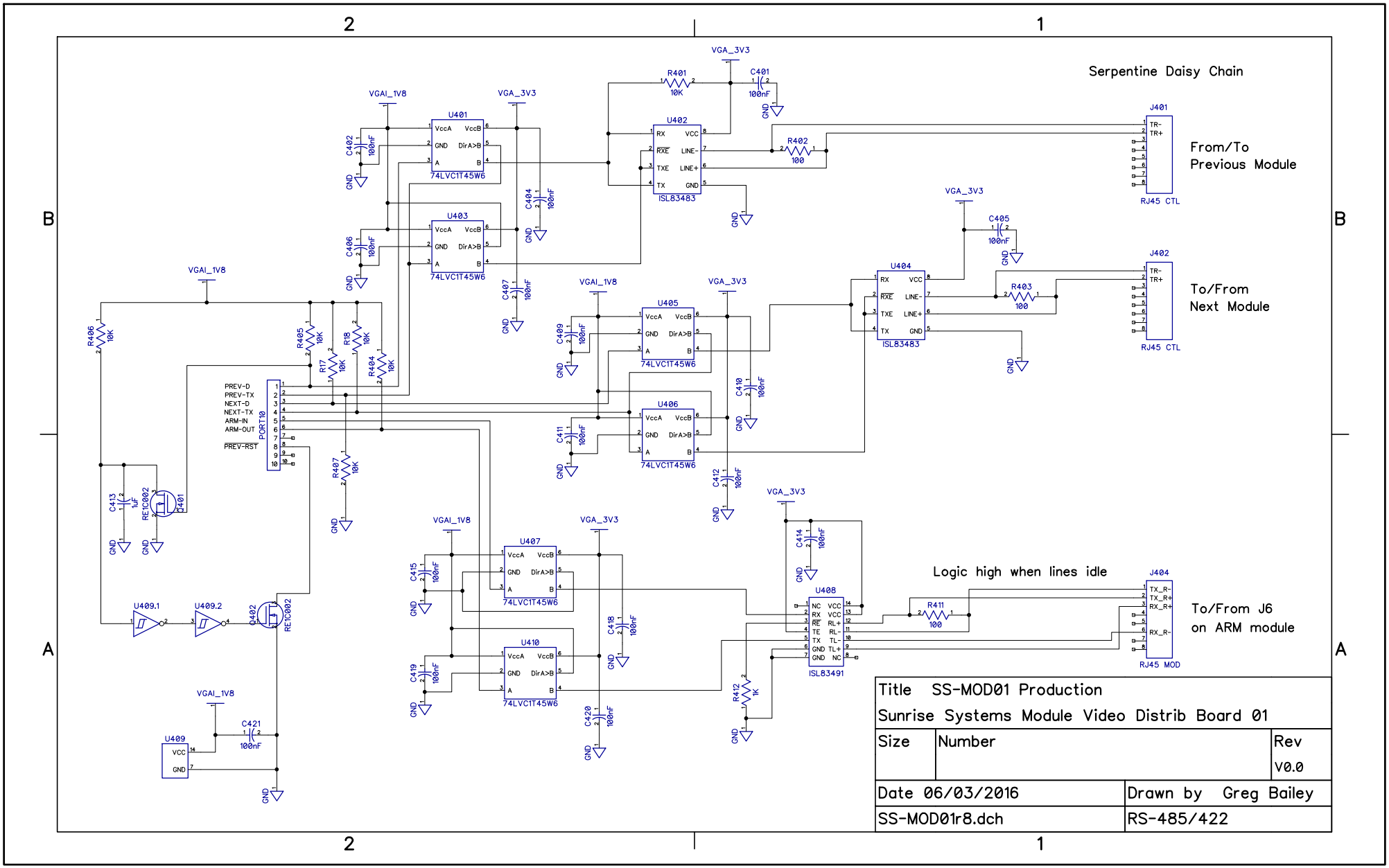
Do we need resistors between input pins and ground or can they go directly to ground?

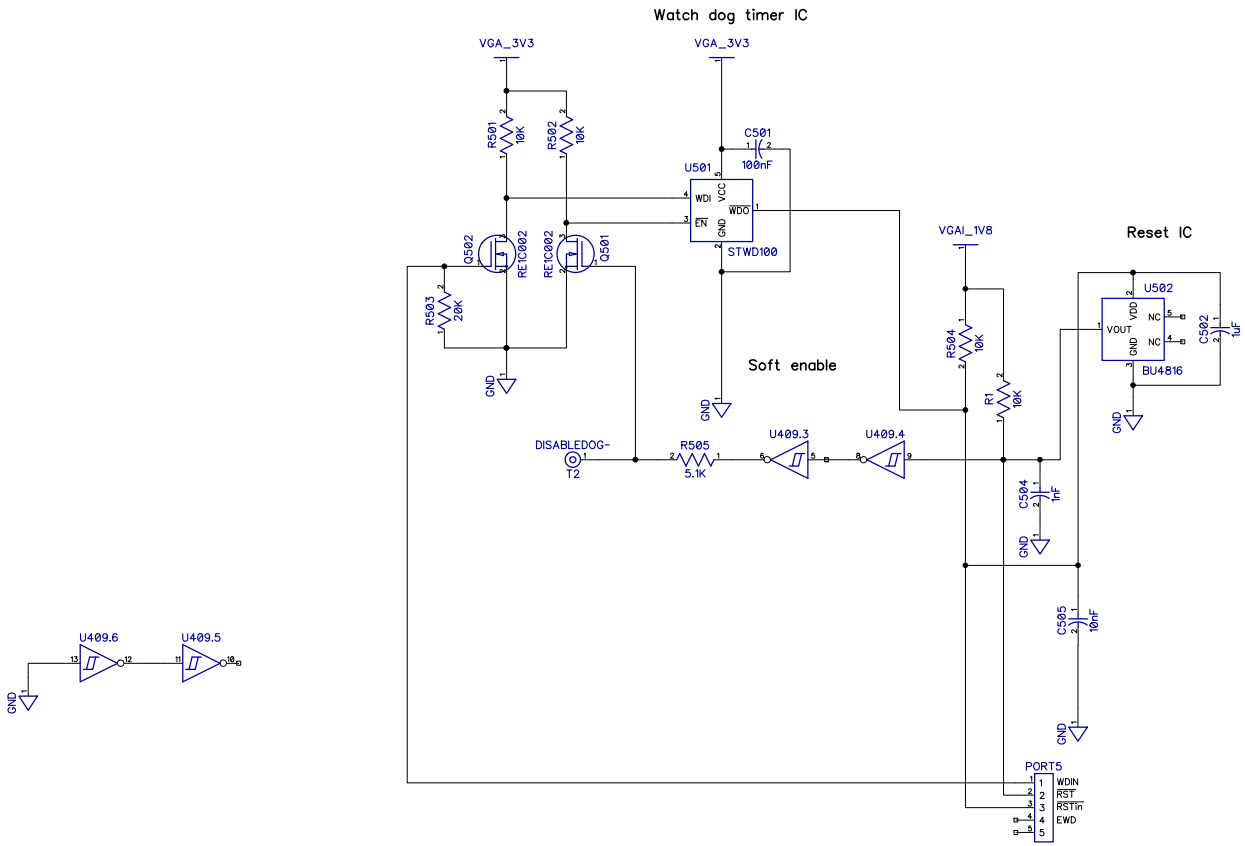
Title SS-MOD01 Production		
Sunrise Systems Module Video Distrib Board 01		
Size	Number	Rev V0.0
Date 06/03/2016	Drawn by Greg Bailey	
SS-MOD01r8.dch	GA144	



Title SS-MOD01 Production Sunrise Systems Module Video Distrib Board 01		
Size	Number	Rev V0.0
Date 06/03/2016	Drawn by Greg Bailey	
SS-MOD01r8.dch	Power	


Video OUT
To Module Below
J301
CLK+
CLK-
D2+
D1+
D1-
D2-
D0+
D0-
Video OUT
PORT8
R301
R303
R305
R307
R308
R311
R313
R317
R318
R321
R323
R326
100
200
Video IN
From Module Above
J302
Video IN
PORT4
VGA_3V3
SN65LVDS33D
U301
R309
R316
R319
R327
R331
R332
R333
453
562
R324
1K
C309
100nF
C302
C304
C312
90pF
C301
C303
C307
C311
1 nF
R302
R304
R306
R310
R312
R315
R320
R322
R325
R328
R329
R330
GND
0F caps leave empty
and pass trace thru pad
ground close as possible to other pad
Title SS-MOD01 Production
Sunrise Systems Module Video Distrib Board 01
Size
Number
Rev
V0.0
Date 06/03/2016
Drawn by Greg Bailey
SS-MOD01r8.dch
Pixel Pipeline


Serpentine Daisy Chain
From/To
Previous Module
To/From
Next Module
Logic high when lines idle
To/From J6
on ARM module
Title SS-MOD01 Production
Sunrise Systems Module Video Distrib Board 01
Size
Number
Rev
V0.0
Date 06/03/2016
Drawn by Greg Bailey
SS-MOD01r8.dch
RS-485/422



Title SS-MOD01 Production		
Sunrise Systems Module Video Distrib Board 01		
Size	Number	Rev V0.0
Date 06/03/2016	Drawn by Greg Bailey	
SS-MOD01r8.dch	Reset	

6. Revision History

REVISION	DESCRIPTION
150908	First Draft.
150917	Include input from Eric and crew. Include results of 2-wire study and the working model of parallel daisy chain with pick-off capability.
150924	Update scope trace for overall parallel pass-through delay with output probe properly grounded. Add list of deferred matters. Update table of module geometries. Add initial FPGA specification and update open questions. Add preliminary system block diagram.
151119	Update block diagram. Add decision to provide control channel. Add information about module serial communications. Add partial description of control board and detailed description of the module board as of version SS-MOD1r2
151228	Replace old descriptions with System Design section, update to describe the prototype.
151229	Add Test Plan section.
160313	Lessons Learned for the VMOD prototype. Software Description (protocols and VMOD software).
160616	Extensive description of VCTL and VMOD, lessons learned from both. Document production VMOD changes. Document final architecture and code behavior for production VMOD.
160628	Document sign configuration language and configuration management procedure.
171104	Reconfigure and sanitize as an application note. Pre-publication draft.
171106	Release for Publication.

GreenArrays®

Application Note AN020 *Revised 11/06/17*

IMPORTANT NOTICE

GreenArrays Incorporated (GAI) reserves the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to GAI's terms and conditions of sale supplied at the time of order acknowledgment.

GAI disclaims any express or implied warranty relating to the sale and/or use of GAI products, including liability or warranties relating to fitness for a particular purpose, merchantability, or infringement of any patent, copyright, or other intellectual property right.

GAI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using GAI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

GAI does not warrant or represent that any license, either express or implied, is granted under any GAI patent right, copyright, mask work right, or other GAI intellectual property right relating to any combination, machine, or process in which GAI products or services are used. Information published by GAI regarding third-party products or services does not constitute a license from GAI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from GAI under the patents or other intellectual property of GAI.

Reproduction of GAI information in GAI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. GAI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of GAI products or services with statements different from or beyond the parameters stated by GAI for that product or service voids all express and any implied warranties for the associated GAI product or service and is an unfair and deceptive business practice. GAI is not responsible or liable for any such statements.

GAI products are not authorized for use in safety-critical applications (such as life support) where a failure of the GAI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of GAI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by GAI. Further, Buyers must fully indemnify GAI and its representatives against any damages arising out of the use of GAI products in such safety-critical applications.

GAI products are neither designed nor intended for use in military/aerospace applications or environments unless the GAI products are specifically designated by GAI as military-grade or "enhanced plastic." Only products designated by GAI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of GAI products which GAI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

GAI products are neither designed nor intended for use in automotive applications or environments unless the specific GAI products are designated by GAI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, GAI will not be responsible for any failure to meet such requirements.

For current information on GreenArrays products and application solutions, see www.GreenArrayChips.com

Mailing Address: GreenArrays, Inc., 821 East 17th Street, Cheyenne, Wyoming 82001
Printed in the United States of America
Phone (775) 298-4748 fax (775) 548-8547 email Sales@GreenArrayChips.com